



Open Client™ and Open Server™ Common Libraries Reference Manual

**Common Libraries
12.5**

DOCUMENT ID: 32850-01-1250-01

LAST REVISED: May 2001

Copyright © 1989-2001 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase, the Sybase logo, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-FORMS, APT-Translator, APT-Library, Backup Server, ClearConnect, Client-Library, Client Services, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, E-Anywhere, E-Whatever, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Financial Fusion, Financial Fusion Server, Gateway Manager, ImpactNow, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MySupport, Net-Gateway, Net-Library, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RW-DisplayLib, RW-Library, S-Designer, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, Transact-SQL, Translation Toolkit, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server and XP Server are trademarks of Sybase, Inc. 3/01

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Contents

About This Book	vii	
CHAPTER 1	Introducing CS-Library	1
	CS-Library overview	1
	Using CS-Library	2
	Open Client and Open Server applications	2
	A standalone CS-Library application	2
	Structures	3
	The CS_CONTEXT structure	3
	Datatypes, constants, and conventions	4
	Error handling	4
	Two methods of handling messages	4
	Using a callback to handle messages	5
	Inline message handling	7
CHAPTER 2	CS-Library Routines	9
	CS-Library routines	9
	cs_calc	10
	cs_cmp	12
	cs_config	13
	cs_conv_mult	23
	cs_convert	25
	cs_ctx_alloc	32
	cs_ctx_drop	35
	cs_ctx_global	37
	cs_diag	38
	cs_dt_crack	43
	cs_dt_info	45
	cs_loc_alloc	50
	cs_loc_drop	51
	cs_locale	52
	cs_manage_convert	58
	cs_objects	64

cs_prop_ssl_localid.....	70
cs_set_convert.....	71
cs_setnull.....	75
cs_strbuild.....	77
cs_strcmp.....	79
cs_time.....	82
cs_validate_cb.....	84
cs_will_convert.....	85

CHAPTER 3

Bulk-Library.....	89
Overview of Bulk-Library.....	89
Client-side and server-side routines.....	89
Header files.....	90
Linking with Bulk-Library.....	90
The CS_BLKDESC structure.....	91
Bulk-Library client programming.....	91
Bulk-Copy-In operations.....	92
Bulk-Copy-Out operations.....	96
Copying to and from Secure SQL Server.....	98
Bulk-Library gateway programming.....	98
Inside the SRV_LANGUAGE event handler.....	100
Inside the SRV_BULK event handler.....	101
Example.....	102

CHAPTER 4

Bulk-Library Routines.....	103
List of Bulk-Library routines.....	103
blk_alloc.....	104
blk_bind.....	107
blk_colval.....	117
blk_default.....	119
blk_describe.....	120
blk_done.....	123
blk_drop.....	126
blk_getrow.....	128
blk_gettext.....	129
blk_init.....	131
blk_props.....	134
blk_rowalloc.....	138
blk_rowdrop.....	139
blk_rowxfer.....	140
blk_rowxfer_mult.....	143
blk_sendrow.....	147
blk_sendtext.....	148

blk_srvinit	150
blk_textxfer	151
Index	157

About This Book

This book, the *Open Client and Open Server Common Libraries Reference Manual*, contains reference information regarding:

- The C version of CS-Library, which contains utility routines that are useful to both Open Client™ Client-Library™ and Open Server™ Server-Library applications.
- The C version of Bulk-Library, which provides bulk copy routines for Client-Library and Server-Library applications. Bulk copy allows high-speed transfer of data between a database table and program variables.

Note Bulk-Library was referred to in previous Open Client/Server™ releases as “the Bulk Copy routines.”

Audience

This manual is designed to serve as a reference manual for programmers who are writing Client-Library or Open Server applications. It is written for application programmers who are familiar with the C programming language.

How to use this book

When writing an Open Client or Open Server application, use the *Common Libraries Reference Manual* as a source of reference information for CS-Library and Bulk-Library routines.

- Chapter 1, “Introducing CS-Library” contains a brief introduction to CS-Library.
- Chapter 2, “CS-Library Routines” contains specific information about each CS-Library routine, such as what parameters the routine takes and what it returns.
- Chapter 3, “Bulk-Library” contains a brief introduction to Bulk-Library.
- Chapter 4, “Bulk-Library Routines” contains specific information on each Bulk-Library routine.

Related documents

- The *Open Client Client-Library Programmer's Guide* contains information on how to design and implement Client-Library programs.
- The *Open Client Client-Library Reference Manual* contains reference information for Client-Library.
- The *Open Server Server-Library Reference Manual* contains reference information for Server-Library.
- The *Open Client/Server Programmer's Supplement* contains platform-specific material needed by developers who use the Open Client/Server products. This document includes information about:
 - Compiling and linking an application
 - The example programs that are included online with Open Client/Server products
 - Routines that have platform-specific behavior
- The *Open Client/Server Configuration Guide* contains information needed by system administrators who configure the Open Client/Server installation environment. This document includes information about:
 - Platform-specific localization mechanisms
 - Configuring Sybase drivers for network services
 - The interfaces file and other configuration files
- The *Open Client/Server International Developer's Guide* contains information needed by programmer's who develop international applications with Client-Library. This document includes:
 - A description of the localization mechanism used by the Open Client and Open Server libraries
 - Guidelines for developing international applications with the Open Client and Open Server libraries

Other sources of information

Use the Sybase Technical Library CD and the Technical Library Product Manuals Web site to learn more about your product:

- Technical Library CD contains product manuals and technical documents and is included with your software. The DynaText browser (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.

- Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you'll find links to the Technical Documents Web site (formerly known as Tech Info Library), the Solved Cases page, and Sybase/Powersoft newsgroups.

To access the Technical Library Product Manuals Web site, go to Product Manuals at <http://www.sybase.com/support/manuals/>.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **For the latest information on product certifications**

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Select a product from the product pick list and click Go.
- 3 Select the Certification Report filter, specify a time frame, and click Go.
- 4 Click a Certification Report title to display the report.

❖ **For the latest information on EBFs and Updates**

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Select EBFs/Updates. Enter user name and password information, if prompted (for existing web accounts) or create a new account (a free service).
- 3 Specify a time frame and click Go.
- 4 Select a product.
- 5 Click an EBF/Update title to display the report.

❖ **To create a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase web pages.

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>
- 2 Click MySybase and create a MySybase profile.

Conventions

CS-Library routine syntax is shown in a bold, monospace font:

```
CS_RETCODE cs_ctx_alloc(version, ctx_pointer)
```

Program text and computer output are shown in a monospace font:

```
cs_ctx_alloc(CS_VERSION_100, &context);
```

Structure names and symbolic constants appear in capital letters (to match their definitions in the *csstypes.h* header file):

```
CS_CONTEXT, CS_EXTRA_INF
```

Routine names and Transact_SQL® keywords are written in a narrow, bold font:

```
cs_ctx_alloc, the select statement
```

Code fragments in this book are taken from the online example programs that are included with Client-Library and Server-Library.

The example programs and the code fragments in this book use *EX_**, *Ex_**, and *ex_** #defines, variables, and routines. These #defines, variables, and routines are part of the example programs, but are not a part of CS-Library, Client-Library, or Server-Library.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

Introducing CS-Library

Topic	Page
CS-Library overview	1
Using CS-Library	2
Structures	3
Error handling	4

CS-Library overview

CS-Library provides utility routines for use in application program development.

CS-Library includes routines to support:

- Datatype conversion
- Arithmetic operations
- Character-set conversion
- Datetime operations
- Sort-order operations
- Localized error messages

CS-Library also includes routines to allocate and deallocate CS-Library structures.

Although you can write a stand-alone CS-Library application, CS-Library's primary function is to provide common utility routines to Client-Library and Server-Library applications.

Because Client-Library and Server-Library programs require a context structure, which can only be allocated using CS-Library, all Client-Library and Server-Library programs include at least two calls to CS-Library—one to allocate a CS_CONTEXT and one to deallocate it.

A context structure contains information about an application's runtime environment, or "context." For more information about the CS_CONTEXT structure, see "Structures" on page 3.

Using CS-Library

You can call CS-Library routines either from within a Client-Library or Server-Library application, or from within a standalone CS-Library application.

Open Client and Open Server applications

Most typically, CS-Library routines are called from within a Client-Library or Server-Library application.

Because the Client-Library and Server-Library header files *ctpublic.h* and *ospublic.h* include the CS-Library header file *cspublic.h*, Client-Library or Server-Library applications do not have to include an additional header file to make CS-Library calls.

After calling `cs_ctx_alloc` to allocate a CS_CONTEXT, a Client-Library or Server-Library application is free to call any other CS-Library routine.

A standalone CS-Library application

It is possible to write a standalone CS-Library application, although this is not a typical use of CS-Library. For example, a standalone application might make CS-Library calls to use the Open Client/Server datatypes and datatype conversion routines.

This type of application needs to include the standard CS-Library header file, *cspublic.h*.

The *Open Client/Server Programmer's Supplement* includes compiling and linking instructions for CS-Library on your platform.

Structures

CS-Library makes use of several structures, including the CS_CONTEXT control structure, the CS_DATAFMT data format structure, and the CS_LOCALE locale information structure.

The CS_CONTEXT structure is a hidden structure whose internals are not available to an application. The CS_CONTEXT is discussed briefly in the following section.

The CS_CONTEXT structure is also required for Client-Library and Server-Library applications.

- For more information about how Client-Library uses the CS_CONTEXT structure, see the *Open Client Client-Library/C Reference Manual* or the *Open Client Client-Library/C Programmer's Guide*.
- For more information about how Server-Library uses the CS_CONTEXT structure, see the *Open Server Server-Library/C Reference Manual*.

The CS_DATAFMT and CS_LOCALE structures are documented in Chapter 2, "Topics," in the *Open Client Client-Library/C Reference Manual*.

The CS_CONTEXT structure

CS-Library defines a single control structure, the CS_CONTEXT.

A CS_CONTEXT structure stores configuration information that describes a particular programming context. An application must allocate a CS_CONTEXT structure before calling any other Client-Library, Server-Library, or CS-Library routine.

An application allocates a CS_CONTEXT structure by calling `cs_ctx_alloc` or `cs_ctx_global`.

An application can customize a CS_CONTEXT by changing the values of context properties. The following routines change the values of context properties:

- The CS-Library routine `cs_config` (after the context has been allocated)
- The Client-Library routine `ct_config` (after the Client-Library routine `ct_init` has been called for the context)
- The Server-Library routine `srv_props` (after calling the Server-Library routine `srv_version` for the context)

An application should deallocate all existing context structures before exiting. An application deallocates a CS_CONTEXT structure by calling `cs_ctx_drop`.

Datatypes, constants, and conventions

CS-Library uses the same datatypes, constants, and conventions as Client-Library and Server-Library. See:

- The “Using Open Client/Server Datatypes” chapter in the *Open Client Client-Library/C Programmer’s Guide*
- The “Types” section in the *Open Client Client-Library/C Reference Manual*
- The “Types” section page in the *Open Server Server-Library/C Reference Manual*

Error handling

All CS-Library routines return success or failure indications. Sybase strongly recommends that applications check these return codes.

In addition, CS-Library routines can generate CS-Library messages, which range in severity from informational messages to fatal errors. Applications should take steps to receive and handle these messages. In most cases, when a CS-Library routine fails, CS-Library generates a message that describes the reason for the failure.

Two methods of handling messages

An application can handle CS-Library messages in one of two ways:

- By installing a callback routine to handle messages
- Inline, using the CS-Library routine `cs_diag`

The callback method has the advantages of:

- Gracefully handling unexpected errors

CS-Library automatically calls the appropriate message callback routine whenever a message is generated, so an application will not fail to trap unexpected errors. An application using only inline error-handling logic may not successfully trap errors that have not been anticipated.

- Centralizing message-handling code

Since all errors are handled in the callback, there is no need to add inline message-handling code after each CS-Library call.

Inline message handling has the advantage of allowing an application to check for messages at particular times. For example, an application that makes a sequence of calls to establish a connection might wait until the connection-related call sequence is complete before checking for messages.

Most applications use the callback method to handle messages.

An application indicates which method it will use for a particular context either by calling `cs_config` to install a message callback routine or by calling `cs_diag` to initialize inline message handling.

An application can switch back and forth between the inline method and the callback method:

- Installing a message callback routine turns off inline message handling. Any saved messages are discarded.
- Likewise, calling `cs_diag` to initialize inline message handling “de-installs” the application’s CS-Library message callback. As a result, the application’s first `CS_GET` call to `cs_diag` will retrieve a warning message to this effect.

If a message callback is not installed and inline message handling is not enabled, CS-Library discards message information.

Using a callback to handle messages

To handle CS-Library errors with a callback function, your application must:

- Declare the callback function as described in “Defining a CS-Library message callback” on page 6.
- Install the callback error handler by calling `cs_config` to set the `CS_MESSAGE_CB` property. For a detailed description, see “CS-Library Message Callback property” on page 20.

Defining a CS-Library message callback

A CS-Library message callback is defined as follows:

```
CS_INT cslibmsg_cb(context, message)

CS_CONTEXT *context;
CS_CLIENTMSG *message;
```

where:

context is a pointer to the CS_CONTEXT structure for which the message occurred.

message is a pointer to a CS_CLIENTMSG structure containing message information. For information on the CS_CLIENTMSG structure, see the “CS_CLIENTMSG Structure” topics page in the *Open Client Client-Library/C Reference Manual*. Note the following similarities with Client-Library:

- Error severities for CS-Library errors have the same meaning as for Client-Library errors.
- The *message->msgnumber* field is a bit-packed CS_INT. This number is unpacked with the macros CS_LAYER, CS_ORIGIN, CS_NUMBER, and CS_SEVERITY. This method is the same for Client-Library messages.

Note that *message* can have a new value each time the message callback is called.

A CS-Library message callback must return either:

- CS_SUCCEED, to instruct CS-Library to continue any processing that is currently occurring on this context, or
- CS_FAIL, to instruct CS-Library to terminate any processing that is currently occurring on this context.

CS-Library message callback example

```
/*
** cslib_err_handler() - CS-Library error handler.
**
** This routine is the CS-Library error handler used by this
** application. It is called by CS-Library whenever an error
** occurs. Here, we simply print the error and return.
**
** Parameters:
** context
**     A pointer to the context handle for context
**     on which the error occurred.
```



```

**      error_msg
**      The structure containing information about the
**      error.
**
** Returns:
**      CS_SUCCEED
*/
CS_RETCODE CS_PUBLIC cslib_err_handler(context, errmsg)
CS_CONTEXT      *context;
CS_CLIENTMSG    *errmsg;
{
    /*
    ** Print the error details.
    */
    fprintf(stdout, "CS-Library error: ");
    fprintf(stdout, "LAYER = (%ld) ORIGIN = (%ld) ",
            CS_LAYER(errmsg->msgnumber),
            CS_ORIGIN(errmsg->msgnumber) );
    fprintf(stdout, "SEVERITY = (%ld) NUMBER = (%ld)\n",
            CS_SEVERITY(errmsg->msgnumber),
            CS_NUMBER(errmsg->msgnumber) );
    fprintf(stdout, "\t%s\n", errmsg->msgstring);
    /*
    ** Print any operating system error information.
    */
    if( errmsg->osstringlen > 0 )
    {
        fprintf(stdout, "CS-Library OS error %ld - %s.\n",
                errmsg->osnumber, errmsg->osstring);
    }
    /*
    ** All done.
    */
    return (CS_SUCCEED);
}

```

Inline message handling

An application calls `cs_diag` to initialize inline CS-Library message handling for a context.

An application that is retrieving messages into `SQLCA`, `SQLCODE`, or `SQLSTATE` must set the CS-Library property `CS_EXTRA_INF` to `CS_TRUE`.

For information on the inline method of handling CS-Library messages, see the reference page for `cs_diag` in Chapter 2, “CS-Library Routines”

This chapter contains a reference page for each CS-Library routine.

CS-Library routines

Routine	Description
cs_calc	Perform an arithmetic operation on two operands.
cs_cmp	Compare two data values.
cs_config	Set or retrieve CS-Library properties.
cs_conv_mult	Retrieve the conversion multiplier for converting character data from one character set to another.
cs_convert	Convert a data value from one datatype, locale, or format to another datatype, locale, or format.
cs_ctx_alloc	Allocate a CS_CONTEXT structure.
cs_ctx_drop	Deallocate a CS_CONTEXT structure.
cs_ctx_global	Allocate or return a CS_CONTEXT structure.
cs_diag	Manage inline error handling.
cs_dt_crack	Convert a machine-readable datetime value into a user-accessible format.
cs_dt_info	Set or retrieve language-specific datetime information.
cs_loc_alloc	Allocate a CS_LOCALE structure.
cs_loc_drop	Deallocate a CS_LOCALE structure.
cs_locale	Load a CS_LOCALE structure with localization values or retrieve the locale name previously used to load a CS_LOCALE structure.
cs_manage_convert	Install or retrieve a user-defined character set conversion routine.
cs_objects	Save, retrieve, or clear objects and data associated with them.
cs_set_convert	Install or retrieve a user-defined conversion routine.

Routine	Description
cs_setnull	Define a null substitution value to be used when binding or converting NULL data.
cs_strbuild	Construct native language message strings.
cs_strcmp	Compare two strings using a specified sort order.
cs_time	Retrieve the current date and time.
cs_will_convert	Indicate whether a specific datatype conversion is available in the Client/Server libraries.

cs_calc

Description

Perform an arithmetic operation on two operands.

Syntax

```
CS_RETCODE cs_calc(context, op, datatype, var1,  
                    var2, dest)
```

```
CS_CONTEXT *context;  
CS_INT      op;  
CS_INT      datatype;  
CS_VOID     *var1;  
CS_VOID     *var2;  
CS_VOID     *dest;
```

Parameters

context

A pointer to a CS_CONTEXT structure.

op

One of the following symbolic values:

Value of <i>op</i>	Arithmetic Operation	*<i>dest</i> Value on Return
CS_ADD	Addition	var1 + var2
CS_SUB	Subtraction	var1 - var2
CS_MULT	Multiplication	var1 * var2
CS_DIV	Division	var1 /var2

datatype

One of following symbolic values, to indicate the datatype of *var1*, *var2*, and *dest*:

Value of <i>datatype</i>	To Indicate This Datatype
CS_DECIMAL_TYPE	CS_DECIMAL
CS_MONEY_TYPE	CS_MONEY
CS_MONEY4_TYPE	CS_MONEY4
CS_NUMERIC_TYPE	CS_NUMERIC

**var1*, **var2*, and **dest* must all be the same datatype as indicated by the value of *datatype*.

var1

A pointer to the first operand for the arithmetic operation.

var2

A pointer to the second operand for the arithmetic operation.

dest

A pointer to a destination buffer. If *cs_calc* returns CS_FAIL, **dest* is not modified.

Return value

cs_calc can return the following values:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a *cs_calc* failure include:

- An invalid parameter
- Attempted division by 0
- Destination overflow

cs_calc generates a CS-Library error message for most failure conditions. For more information on CS-Library error handling, see “Error handling” on page 4.

Usage

- *var1*, *var2*, and *dest* must have the same datatype, as indicated by the *datatype* parameter.
- In case of error, **dest* is not modified.

See also

cs_convert

cs_cmp

Description Compare two data values.

Syntax CS_RETCODE cs_cmp(context, datatype, var1, var2,
result)
CS_CONTEXT *context;
CS_INT datatype;
CS_VOID *var1;
CS_VOID *var2;
CS_INT *result;

Parameters

context

A pointer to a CS_CONTEXT structure.

datatype

One of following symbolic values, to indicate the datatype of *var1* and *var2*:

Value of <i>datatype</i>	To Indicate This Datatype
CS_DATETIME_TYPE	CS_DATETIME
CS_DATETIME4_TYPE	CS_DATETIME4
CS_DECIMAL_TYPE	CS_DECIMAL
CS_MONEY_TYPE	CS_MONEY
CS_MONEY4_TYPE	CS_MONEY4
CS_NUMERIC_TYPE	CS_NUMERIC

var1

A pointer to the first operand for the comparison.

var2

A pointer to the second operand for the comparison.

result

On successful return, **result* is set to indicate the result of the comparison:

Value of <i>*result</i>	To Indicate
-1	<i>var1</i> is less than <i>var2</i> .
0	<i>var1</i> is equal to <i>var2</i> .
1	<i>var1</i> is greater than <i>var2</i> .

Return value

cs_cmp can return the following values:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed. If cs_cmp returns CS_FAIL, <i>*result</i> is undefined.

The most common reason for a `cs_cmp` failure is an invalid parameter.

`cs_cmp` generates a CS-Library error message for most failure conditions. For more information on CS-Library error handling, see “Error handling” on page 4.

Usage

- `cs_cmp` sets **result* to indicate the result of the comparison.
- *var1* and *var2* must have the same datatype, as indicated by the *datatype* parameter.
- To compare string values, an application can call `cs_strcmp`.

See also

`cs_calc`, `cs_convert`, `cs_strcmp`

cs_config

Description

Set or retrieve CS-Library properties.

Syntax

```
CS_RETCODE cs_config(context, action, property,
                    buffer, buflen, outlen)
```

```
CS_CONTEXT    *context;
CS_INT        action;
CS_INT        property;
CS_VOID       *buffer;
CS_INT        buflen;
CS_INT        *outlen;
```

Parameters

context

A pointer to a `CS_CONTEXT` structure.

action

One of the following symbolic values:

<i>action</i>	<i>cs_config</i>
<code>CS_SET</code>	Sets the value of the property.
<code>CS_GET</code>	Retrieves the value of the property.
<code>CS_CLEAR</code>	Clears the value of the property by resetting it to its default value.

property

The property whose value is being set or retrieved, according to the following table:

Table 2-1: Values for cs_config's property parameter

Value of <i>property</i>	Controls	Action	*<i>buffer</i> Is
CS_APPNAME	The name the application calls itself.	Set, retrieve, or clear.	A CS_CHAR string. The default is NULL.
CS_CONFIG_FILE	The name and path of the Open Client/Server runtime configuration file. Meaningful only when external configuration has been enabled by setting CS_EXTERNAL_CONFIG.	Set, retrieve, or clear.	A CS_CHAR string. The default is NULL, which means a platform-specific default is used. See "Configuration file property" on page 18 for more information.
CS_EXTERNAL_CONFIG	Whether or not the Client-Library routine ct_init reads an external configuration file to set default property values.	Set, retrieve, or clear.	CS_TRUE or CS_FALSE. The default depends on whether the external configuration file exists. See "External configuration property" on page 18 for more information.
CS_EXTRA_INF	Whether or not to return the extra information that is required when processing messages inline using a SQLCA, SQLCODE, or SQLSTATE structure.	Set, retrieve, or clear.	CS_TRUE or CS_FALSE. CS_FALSE is the default.
CS_LOC_PROP	A CS_LOCALE structure that defines localization information for this context.	Set, retrieve, or clear.	A CS_LOCALE structure previously allocated by the application.

Value of <i>property</i>	Controls	Action	<i>*buffer</i> Is
CS_MESSAGE_CB	The CS-Library message callback routine, which is an application-provided handler for CS-Library error and informational messages.	Set, retrieve, or clear.	If <i>action</i> is CS_SET, <i>*buffer</i> is the message callback routine. If <i>action</i> is CS_GET, <i>*buffer</i> is set to the address of the message callback routine that is currently installed. The default is NULL, which means no handler is installed.
CS_NOAPI_CHK	Whether or not CS-Library validates function arguments when library functions are called.	Set, retrieve, or clear.	CS_TRUE or CS_FALSE. CS_FALSE indicates that argument checking is performed. CS_FALSE is the default.
CS_USERDATA	User-allocated data.	Set, retrieve, or clear.	User-allocated data. A default is not applicable.

Value of <i>property</i>	Controls	Action	<i>*buffer</i> Is
CS_VERSION	The version of CS-Library.	Retrieve only.	A symbolic code indicating the library version: <ul style="list-style-type: none">• CS_VERSION_100 indicates the context exhibits version 10.0 behavior.• CS_VERSION_110 indicates version 11.1 behavior.• CS_VERSION_120 indicates the context exhibits version 12.0 behavior.• CS_VERSION_125 indicates version 12.5 behavior.

buffer

If a property value is being set, *buffer* points to the value to use in setting the property.

If a property value is being retrieved, *buffer* points to the space in which *cs_config* will place the value of the property.

If a property value is being cleared, pass *buffer* as NULL and *buflen* as CS_UNUSED.

buflen

Generally, *buflen* is the length, in bytes, of **buffer*.

If a property value is being set and the value in **buffer* is null-terminated, pass *buflen* as CS_NULLTERM.

If **buffer* is a fixed-length or symbolic value, pass *buflen* as CS_UNUSED.

outlen

A pointer to an integer variable.

outlen is not used if a property value is being set.

If a property value is being retrieved, `cs_config` sets **outlen* to the length, in bytes, of the requested information.

If the information is larger than *buflen* bytes, an application can use the value of **outlen* to determine how many bytes are needed to hold the information.

outlen can be passed as NULL if the application is setting a property value or does not require the output length of a retrieved value.

Return value

cs_config returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Usage

- There are three kinds of context properties:
 - Context properties specific to CS-Library
 - Context properties specific to Client-Library
 - Context properties specific to Server-Library

`cs_config` sets and retrieves the values of CS-Library context properties. With the exception of `CS_LOC_PROP`, properties set via `cs_config` affect only CS-Library.

`ct_config` sets and retrieves the values of Client-Library-specific context properties. Properties set via `ct_config` affect only Client-Library.

`srv_props` sets and retrieves the values of Server-Library-specific context properties. Properties set via `srv_props` affect only Server-Library.

- See the “Properties” topics page in the *Open Client Library/C Reference Manual* for information about Client-Library properties.

Application name property

- `CS_APPNAME` specifies the name that the application calls itself.
- `cs_config` sets the application name for a `CS_CONTEXT` structure. In a Client-Library application, allocated connections inherit the application name from their parent `CS_CONTEXT` structure.

- The application name specifies a section name in the Open Client/Server runtime configuration file. See “Configuration file property” on page 18 for more information.
- CS_APPNAME cannot be set, retrieved, or cleared unless the CS_CONTEXT structure was allocated with CS_VERSION_110 or later.

Configuration file property

- CS_CONFIG_FILE specifies the name and path to the Open Client/Server runtime configuration file.
- The default value is NULL, which means that the a platform-specific default file will be used:
 - On UNIX platforms, the default file is `$$SYBASE/SYBASE_OCS/ocs.cfg` where `$$SYBASE` is the path to the Sybase installation directory; this path is specified as the value of the SYBASE environment variable.
 - On Windows platforms, the default file is `%SYBASE%\SYBASE_OCS\ocs.cfg`, where `%SYBASE%` is the path to the Sybase installation directory; this path is specified as the value of the SYBASE environment variable.
 - For other platforms, see the *Open Client/Server Configuration Guide* for the name of the default Open Client/Server runtime configuration file.

The *Open Client/Server Configuration Guide* describes the structure of the Sybase installation directory.

- If the default external-configuration file exists, Client-Library reads configuration settings from it unless the application explicitly sets the CS_EXTERNAL_CONFIG property to CS_FALSE. See “External configuration property” on page 18.
- CS_CONFIG_FILE cannot be set, retrieved, or cleared unless the CS_CONTEXT structure was allocated with CS_VERSION_110 or later.

External configuration property

- CS_EXTERNAL_CONFIG controls whether the Client-Library routine `ct_init` will read the Open Client/Server runtime configuration file to set default Client-Library property values for the CS_CONTEXT structure.
- The name of the Open Client/Server runtime configuration file is specified with the CS_CONFIG_FILE property. See “Configuration file property” on page 18.

- The default for CS_EXTERNAL_CONFIG depends on whether the default external-configuration file exists (see “Configuration file property” on page 18). If the default external-configuration file exists, then CS_EXTERNAL_CONFIG defaults to CS_TRUE. Otherwise, CS_EXTERNAL_CONFIG defaults to CS_FALSE.
- Configuration information is read from the section of the file labeled:
`[appname]`
 where *appname* is the value of the CS_APPNAME property. (See “Application name property” on page 17.) If the application has not set the CS_APPNAME property, the configuration reads the section labeled:
`[DEFAULT]`
- The “Using the Open Client/Server Runtime Configuration File” topics page in the *Open Client Client-Library/C Reference Manual* describes the syntax and keywords for configuration file entries.
- CS_EXTERNAL_CONFIG cannot be set, retrieved, or cleared unless the CS_CONTEXT structure is allocated with CS_VERSION_110 or later. (See `cs_ctx_alloc` for more information.)

Extra Information property

- CS_EXTRA_INF determines whether or not CS-Library returns the extra information that is required to fill in a SQLCA, SQLCODE, or SQLSTATE structure.
- If an application is not retrieving messages into a SQLCA, SQLCODE, or SQLSTATE structure, the extra information is returned as ordinary CS-Library messages.

Locale information property

- The CS_LOC_PROP property defines a CS_LOCALE structure that contains localization information for a context. Localization information includes a language, character set, datetime, money, and numeric formats, and a collating sequence.
- CS_LOC_PROP affects both CS-Library and Client-Library, because a new connection picks up default localization information from its parent context.

- If an application does not call `cs_config` to define localization information for a context, the context uses default localization information that it picks up from the operating system environment when it is allocated. If localization information is not available in the operating system environment, the context uses platform-specific default localization values.
- The routine `cs_loc_alloc` allocates a `CS_LOCALE` structure.

CS-Library Message Callback property

- The `CS_MESSAGE_CB` property consists of a pointer to a user-supplied CS-Library message callback routine. The application uses this property to install a handler for error or informational messages from CS-Library.
 - The default value is `NULL`, meaning that no handler is installed.
 - An application function can be installed as a handler for CS-Library errors.
 - Once the handler is installed, CS-Library calls the handler when an error or exception occurs in a CS-Library routine.
- For a description and an example of coding a CS-Library error handler, see “Defining a CS-Library message callback” on page 6.
- The code fragment below demonstrates how a handler function is installed for CS-Library errors.

```
/*
** Install the function cslib_err_handler as the
** handler for CS-Library errors.
*/
if (cs_config(context, CS_SET, CS_MESSAGE_CB,
              (CS_VOID *)cslib_err_handler,
              CS_UNUSED, NULL)
    != CS_SUCCEEDED)
{
    /* Release the context structure. */
    (void)cs_ctx_drop(context);
    fprintf(stdout,
            "Can't install CS-Lib error handler.\n"
            "Exiting.\n");
    exit(1);
}
```

- Client-Library applications that call CS-Library routines besides `cs_ctx_alloc` and `cs_ctx_drop` need dedicated CS-Library error handling. Applications should either install a CS-Library error handler or handle CS-Library errors inline with `cs_diag`.

Note CS-Library error messages are not sent to the Client-Library error handler.

- Callback error handlers for Client-Library and CS-Library are installed differently:
 - An application installs Client-Library callback routines by calling `ct_callback`.
 - An application installs a CS-Library message callback routine by calling `cs_config` to set the value of the `CS_MESSAGE_CB` property.

Aside from this difference, the CS-Library message callback is similar to the Client-Library client message callback. For general information on callback routines, see the “Callbacks” topics page in the *Open Client Client-Library/C Reference Manual*.

Argument checking for CS-Library calls

- The `CS_NOAPI_CHK` property determines whether or not CS-Library validates function arguments when a library function is called.
- If the value of `CS_NOAPI_CHK` is `CS_FALSE` (the default), then CS-Library checks arguments when API functions are called. Setting `CS_NOAPI_CHK` to `CS_TRUE` disables API checking.
- For argument checking, CS-Library validates the parameters passed with each function call. Pointers to CS-Library hidden structures such as `CS_LOCALE` are checked. Field values in structures are also checked for illegal combinations. If CS-Library finds invalid arguments and API checking is enabled, CS-Library generates error messages and the function will fail. These messages can be trapped and displayed with a CS-Library callback error handler.

- If the value of CS_NOAPI_CHK is CS_TRUE, then arguments are not validated before they are used. If the application passes invalid arguments to CS-Library, then the application will not work right, resulting in memory corruption, memory access violations (UNIX “core dumps”), or incorrect results. No error messages are generated to warn the application of the condition. API argument checking should not be disabled until the application has been completely debugged with API checking enabled.

Warning! Do not set CS_NOAPI_CHK to CS_TRUE unless your application has been completely debugged with the default setting (CS_FALSE).

User-allocated Data property

- The CS_USERDATA property defines user-allocated data. This property allows an application to associate user data with a particular context structure.
- CS-Library copies the user data into internal data space. An application can then call cs_config at a later time to retrieve the data.
- If you do not include a string’s null terminator when calculating its length during the input stage, a call to cs_config (CS_GET) will return only the string (minus its null terminator). For example, if you input a 2-byte string with a null terminator, and specify the string’s length as 2 bytes, cs_config (CS_GET) will return only the string. If, on the other hand, you input a 2-byte string with a null terminator and specify the string’s length as 3 bytes, cs_config (CS_GET) will return the string and its null terminator.
- Although Client-Library also has a CS_USERDATA property, the Client-Library CS_USERDATA is set only at the connection and command levels.

Version level property

- The CS_VERSION property represents the version of CS-Library behavior that an application has requested via cs_ctx_alloc.
- An application can only retrieve the value of CS_VERSION.
- Possible values for CS_VERSION include the following:
 - CS_VERSION_100 indicates version 10.0 behavior
 - CS_VERSION_110 indicates version 11.1 behavior
 - CS_VERSION_120 indicates version 12.0 behavior

- CS_VERSION_125 indicates version 12.5 behavior

See also `cs_ctx_alloc`, `ct_con_props`, `ct_config`, `ct_init`

cs_conv_mult

Description Retrieve the conversion multiplier for converting character data from one character set to another.

Syntax

```
CS_RETCODE cs_conv_mult(context,
                        srcloc,
                        destloc,
                        conv_multiplier)
CS_CONTEXT *context;
CS_LOCALE *srcloc;
CS_LOCALE *destloc;
CS_INT *conv_multiplier;
```

Parameters

context
A pointer to a CS_CONTEXT structure.

srcloc
A pointer to the CS_LOCALE structure that describes the source variable's character set. This parameter cannot be NULL.

destloc
A pointer to the CS_LOCALE structure that describes the destination variable's character set. This parameter cannot be NULL.

conv_multiplier
A pointer to a CS_INT variable. `cs_conv_mult` retrieves the conversion multiplier for conversions from the character set indicated by *srcloc* to the character set indicated by *destloc* and places it into **conv_multiplier*.

Return value `cs_conv_mult` returns the following values:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a `cs_conv_mult` failure is an invalid parameter.

Examples The following code fragment retrieves the conversion multiplier for conversions from the iso_1 character set to the eucjis character set:

```
#define EXIT_ON_FAIL(context, ret, msg) \
```

```
{ if (ret != CS_SUCCEED) \
  { \
    fprintf(stdout, "Fatal error(%ld): %s\n", (long)ret, msg); \
    if (context != (CS_CONTEXT *)NULL) \
      { (CS_VOID)ct_exit(context, CS_FORCE_EXIT); \
        (CS_VOID)cs_ctx_drop(context); } \
    exit(-1); \
  } }

** usa_locale uses the iso_1 character set.
*/
ret = cs_loc_alloc(context, &usa_locale);
EXIT_ON_FAIL(context, ret, "cs_loc_alloc(usa) failed.");
ret = cs_locale(context, CS_SET, usa_locale,
                CS_SYB_CHARSET, "iso_1", CS_NULLTERM, NULL);
EXIT_ON_FAIL(context, ret, "cs_locale(usa, CHARSET) failed.");
/*
** japan_locale uses eucjis.
*/
ret = cs_loc_alloc(context, &japan_locale);
EXIT_ON_FAIL(context, ret, "cs_loc_alloc(japan) failed.");
ret = cs_locale(context, CS_SET, japan_locale,
                CS_SYB_CHARSET, "eucjis", CS_NULLTERM, NULL);
EXIT_ON_FAIL(context, ret, "cs_locale(japan, CHARSET) failed.");
/*
** Get the conversion multiplier for iso_1 to eucjis conversions.
*/
ret = cs_conv_mult(context,
                   usa_locale, japan_locale, &conv_mult);
EXIT_ON_FAIL(context, ret, "cs_conv_mult(usa, japan) failed.");
fprintf(stdout,
        "Conversion multiplier for iso_1 to eucjis is %ld.\n",
        (long)conv_mult);
```

Usage

- **cs_conv_mult** retrieves the conversion multiplier for converting character data from one character set to another.
- The conversion multiplier allows an application to size the destination data space for conversion of character data into a different character set.

- When converted to another character set, character strings can grow or shrink in length, and applications need to make sure that the destination data space is large enough for the result. With a multi-byte character set destination, one byte in the source can convert to several bytes in the destination. Also, when converting to a single-byte character set, some characters may convert to multi-character *mnemonics*. For example, if the destination character set does not contain a character for TM (the trademark symbol), it might convert to the 2-character mnemonic “TM”.
- A *conversion multiplier* equals the largest number of bytes in the destination that can replace 1 source byte.
- When converting a character string to a different character set, the application uses the conversion multiplier to size the destination data space, as follows:

```
bytes_needed = conv_mult
               * srclen
               * CS_SIZEOF(CS_BYTE)
               + NTB
```

where:

bytes_needed is the necessary length, in bytes, of the destination data space.

conv_mult is the the conversion multiplier value.

srclen is the length, in bytes, of the source string.

NTB is 1 if null termination is requested and 0 otherwise.

- For more information on character set conversion, see the *Open Client/Server International Developer's Guide*.

See also

`cs_convert`, `cs_locale`, `cs_manage_convert`

cs_convert

Description	Convert a data value from one datatype, locale, or format to another datatype, locale, or format.
Syntax	<pre>CS_RETCODE cs_convert(context, srcfmt, srcdata, destfmt, destdata, resultlen) CS_CONTEXT *context; CS_DATAFMT *srcfmt; CS_VOID *srcdata;</pre>

CS_DATAFMT *destfmt;
 CS_VOID *destdata;
 CS_INT *resultlen;

Parameters

context

A pointer to a CS_CONTEXT structure.

srcfmt

A pointer to a CS_DATAFMT structure describing the source data format. The fields in **srcfmt* are used as follows:

Field Name	Set It To
<i>datatype</i>	A type constant representing the type of the source data (CS_CHAR_TYPE, CS_BINARY_TYPE, and so on).
<i>maxlength</i>	The length of the data in the <i>*srcdata</i> buffer. This value is ignored for fixed-length datatypes or if <i>srcdata</i> is NULL.
<i>locale</i>	A pointer to a CS_LOCALE structure containing localization values for the source data, or NULL to use localization values from <i>*context</i> .
All other fields	Are ignored.

For general information on the CS_DATAFMT structure, see the “CS_DATAFMT Structure” topics page in the *Open Client Client-Library/C Reference Manual*.

srcdata

A pointer to the source data. To indicate that the source data represents a null value, pass *srcdata* as NULL. If *srcdata* is NULL, *cs_convert* places the null substitution value for the datatype indicated by *destfmt->datatype* in **destdata*.

Table 2-15 on page 76 lists the default null substitution value for each datatype. An application can define custom null substitution values by calling *cs_setnull*.

destfmt

A pointer to a CS_DATAFMT structure describing the destination data format. The following table lists the fields in **destfmt* that are used.

Table 2-2: CS_DATAFMT fields for cs_convert's *destfmt parameter

Field Name	Set It To
<i>datatype</i>	A type constant representing the desired destination datatype (CS_CHAR_TYPE, CS_BINARY_TYPE, and so on).
<i>maxlength</i>	The length of the <i>destdata</i> buffer.
<i>locale</i>	A pointer to a CS_LOCALE structure containing localization values for the destination data, or NULL to use localization values from <i>*context</i> .
<i>format</i>	A bit mask of the following symbols: <ul style="list-style-type: none"> • For character and text destinations only, use CS_FMT_NULLTERM to null-terminate the data, or CS_FMT_PADBLANK to pad to the full length of the variable with spaces. • For character, binary, text, and image destinations, use CS_FMT_PADNULL to pad to the full length of the variable with nulls. • When converting from a character source to a character destination, use CS_FMT_SAFESTR to double any occurrences of the single-quote character (') in the destination. CS_FMT_SAFESTR can be combined with CS_FMT_NULLTERM, CS_FMT_PADBLANK, or CS_FMT_PADNULL. • For any type of destination, use CS_FMT_UNUSED if no format information is being provided.
<i>scale</i>	The scale to be used for the destination variable. If the source data is the same type as the destination, then <i>scale</i> can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for <i>scale</i> from the source data. <i>scale</i> must be less than or equal to <i>precision</i> .
<i>precision</i>	The precision to be used for the destination variable. If the source data is the same type as the destination, then <i>precision</i> can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for <i>precision</i> from the source data. <i>precision</i> must be greater than or equal to <i>scale</i> .
All other fields	Are ignored.

For general information on the CS_DATAFMT structure, see the “CS_DATAFMT Structure” topics page in the *Open Client Client-Library/C Reference Manual*.

destdata

A pointer to the destination buffer space.

resultlen

A pointer to an integer variable. `cs_convert` sets **resultlen* to the length, in bytes, of the data placed in **destdata*. If the conversion fails, `cs_convert` sets **resultlen* to `CS_UNUSED`.

resultlen is an optional parameter and can be passed as `NULL`.

Datatype Conversion Chart

The chart in Table 2-3 indicates which datatype conversions are supported by `cs_convert`. The source datatypes are listed in the leftmost column and the destination datatypes are listed in the top row of the chart. “X” indicates that the conversion is supported; a blank space indicates that the conversion is not supported.

Table 2-3: Datatype conversion chart

Open Client Datatypes

	CS_BINARY	CS_LONGBINARY	CS_VARBINARY	CS_BIT	CS_CHAR	CS_LONGCHAR	CS_VARCHAR	CS_DATETIME	CS_DATETIME4	CS_TINYINT	CS_SMALLINT	CS_INT	CS_DECIMAL	CS_NUMERIC	CS_FLOAT	CS_REAL	CS_MONEY	CS_MONEY4	CS_BOUNDARY	CS_SENSITIVITY	CS_TEXT	CS_IMAGE	
CS_BINARY	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X	X	
CS_LONGBINARY	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X	X
CS_VARBINARY	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X	X
CS_BIT	X	X	X	X	X	X	X			X	X	X	X	X	X	X						X	X
CS_CHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CS_LONGCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CS_VARCHAR	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
CS_DATETIME			X		X	X	X	X															X
CS_DATETIME4			X		X	X	X	X															X
CS_TINYINT	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X				X	X
CS_SMALLINT	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X				X	X
CS_INT	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X				X	X
CS_DECIMAL	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X				X	X
CS_NUMERIC	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X				X	X
CS_FLOAT	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X				X	X
CS_REAL	X	X	X	X	X	X	X			X	X	X	X	X	X	X	X	X				X	X

Open Client Datatypes

CS_MONEY	X X X X X X X	X X X X X X X X	X X
CS_MONEY4	X X X X X X X	X X X X X X X X	X X
CS_BOUNDARY		X X X	X X
CS_SENSITIVITY		X X X	X X
CS_TEXT	X X		
CS_IMAGE	X X X X X X X X X X X X X X X X X X		X X
CS_UNICHAR	X X X X X X X X		

Return value

Returns	To Indicate
CS_SUCCEEDED	The routine completed successfully.
CS_FAIL	The routine failed.

A common reason for a cs_convert failure is that CS-Library does not support the requested conversion.

cs_convert conversion errors will generate CS-Library error messages. For more information about CS-Library error handling, see “Error handling” on page 4.

Usage

- To determine whether a particular conversion is permitted, use cs_will_convert.
- In Client-Library applications, ct_bind sets up automatic, implicit data conversion, which makes it unnecessary for an application to explicitly convert result data that is bound to program variables.
- An application can install custom conversion routines by calling cs_set_convert. Once a custom routine for a particular type of conversion is installed, cs_convert or ct_bind call the custom routine whenever a conversion of that type is required.
- cs_convert can convert between standard and user-defined datatypes. To enable these types of conversions, an application must install the appropriate custom conversion routines via cs_set_convert.

- For more information about CS-Library datatypes, see the “Types” topics page in the *Open Client Library/C Reference Manual*. For information about Adaptive Server datatypes, see the *Adaptive Server Enterprise Reference Manual*.

About specific conversions • A conversion to or from *binary* and *image* datatypes is a straight byte-copy, except when the conversion involves *character* or *text* data. When converting character or text data to binary or image, `cs_convert` interprets the character or text string as hexadecimal, whether or not the string contains a leading “0x”. There must be a match in the lengths of binary data and fixed length data. If the data lengths do not match, there will be underflow or overflow.

- Converting a *money*, *character*, or *text* value to *float* can result in a loss of precision. Converting a float value to character or text can also result in a loss of precision.
- Converting a *float* value to *money* can result in overflow, because the maximum `CS_MONEY` value is \$922,337,203,685,477.5807, and the maximum `CS_MONEY4` value is \$214,748.3648.
- If overflow occurs when converting *integer* or *float* data to *character* or *text*, the first character of the resulting value will contain an asterisk (*) to indicate the error.
- A conversion to *bit* has the following effect: if the value being converted is not 0, the bit value is set to 1; if the value is 0, the bit value is set to 0.
- When converting *decimal* or *numeric* data to decimal or numeric data, `CS_SRC_VALUE` can be used in `destfmt->scale` and `destfmt->precision` to indicate that the destination data should have the same scale and precision as the source. `CS_SRC_VALUE` is valid only when the source data is decimal or numeric.

Note Open Client and Open Server 12.5 support the `unichar` datatype. For information about this datatype, see Chapter 3, “Using Open Client/Server Datatypes”, in the *Open Client Library/C Programmer’s Guide*.

Converting between character sets

- `cs_convert` performs character set conversion when:
 - `srctype` and `desttype` both represent character-based types and
 - `srcfmt->locale` specifies a different character set than `destfmt->locale`.

The character-based types are CS_CHAR, CS_LONGCHAR, CS_TEXT, or CS_VARCHAR.

- You can program an application to perform character-set conversion by following these steps:
 - a Call `cs_loc_alloc` twice to allocate two CS_LOCALE structures, `src_locale` and `dest_locale`, which will be configured to describe the locale of the source data and destination data, respectively.
 - b Configure the character set associated with `src_locale` by calling `cs_locale`. The call can specify either a locale name or a character set name.

To use a character set name, pass *action* as CS_SET, *type* as CS_SYB_CHARSET, and *buffer* as the name of the character set. Repeat to configure the character set for `dest_locale`.

To use a locale name, pass *action* as CS_SET, *type* as CS_LC_CTYPE, and *buffer* as a locale name (the character set associated with the locale name will be used). Repeat to configure the character set for `dest_locale`.

- c (Optional) Call `cs_conv_mult` to get the conversion multiplier for conversions between `src_locale`'s character set and `dest_locale`'s character set. The conversion multiplier can be used to determine whether the result can possibly overflow the destination space.
- d Configure the CS_DATAFMT structures to describe the datatype, length, and format of the source and destination data. Set the *locale* field in the source CS_DATAFMT structure to `src_locale`, and set the locale field in the destination CS_DATAFMT structure to `dest_locale`.
- e Call `cs_convert` to perform the conversion. This step can be repeated as many times as necessary, using the configured CS_LOCALE and CS_DATAFMT structures.
- f Call `cs_loc_drop` to deallocate each of `src_locale` and `dest_locale` when they are no longer needed.

See also

`cs_conv_mult`, `cs_manage_convert`, `cs_set_convert`, `cs_setnull`,
`cs_will_convert`

cs_ctx_alloc

Description

Allocate a CS_CONTEXT structure.

Syntax CS_RETCODE cs_ctx_alloc(version, ctx_pointer)

CS_INT version;
 CS_CONTEXT **ctx_pointer;

Parameters

version

One of the following symbolic values, to indicate the intended version of CS-Library behavior:

Value of <i>version</i>	To Indicate	Features Supported
CS_VERSION_100	10.0 behavior.	Initial version.
CS_VERSION_110	11.1 behavior.	Unicode character set support. Use of external configuration files to control Client-Library property settings.
CS_VERSION_120	12.0 behavior.	All above features.
CS_VERSION_125	12.5 behavior.	unichar support, wide data and columns, SSL.

ctx_pointer

The address of a pointer variable. cs_ctx_alloc sets **ctx_pointer* to the address of a newly allocated CS_CONTEXT structure.

In case of error, cs_ctx_alloc sets **ctx_pointer* to NULL.

Return value

cs_ctx_alloc returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_MEM_ERROR	The routine failed because it could not allocate sufficient memory.
CS_FAIL	The routine failed for other reasons.

The most common reason for a cs_ctx_alloc failure is a misconfigured system environment. cs_ctx_alloc must read the locales file that specifies the default localization values for the allocated context. If CS-Library cannot find the locales file or if the locales file is misconfigured, cs_ctx_alloc will fail.

On most systems, the SYBASE environment variable or logical name specifies the location of the locales file. The *Open Client/Server Configuration Guide* describes the environmental configuration required for CS-Library localization values.

Other common reasons for a cs_ctx_alloc failure include:

- Insufficient memory.

- Missing localization files.
- The value of the LANG environment variable does not match an entry in the locales file.

Note On platforms that have a standard error device, `cs_ctx_alloc` prints U.S. English error messages to the standard error device when CS-Library cannot find the locales file. For Windows and other platforms that lack a standard error device, U.S. English error messages are written to a text file called *sybinit.err* in the application's working directory.

Examples

```
/*
** ex_init()
*/

CS_RETCODE CS_PUBLIC
ex_init(context)
CS_CONTEXT*      *context;
{
    CS_RETCODE      retcode;
    CS_INT          netio_type = CS_SYNC_IO;

    /* Get a context handle to use */
    retcode = cs_ctx_alloc(CS_VERSION_125, context);
    if (retcode != CS_SUCCEEDED)
    {
        ex_error("ex_init: cs_ctx_alloc() failed");
        return retcode;
    }

    /* Initialize Open Client */
    ...CODE DELETED.....

    /* Install client and server message handlers */
    ...CODE DELETED.....

    if (retcode != CS_SUCCEEDED)
    {
        ct_exit(*context, CS_FORCE_EXIT);
        cs_ctx_drop(*context);
        *context = NULL;
    }

    return retcode;
}
```

- Usage
- A `CS_CONTEXT` structure, also called a “context structure,” contains information that describes an application context. For example, a context structure contains default localization information and defines the version of CS-Library that is in use.
 - Allocating a context structure is the first step in any Client-Library or Server-Library application.
 - After allocating a `CS_CONTEXT` structure, a Client-Library application typically customizes the context by calling `cs_config` and/or `ct_config` and then creates one or more connections within the context. A Server-Library application can customize a context by calling `cs_config` and `srv_props`.
 - To deallocate a context structure, an application can call `cs_ctx_drop`.
 - `cs_ctx_global` also allocates a context structure. The difference between `cs_ctx_alloc` and `cs_ctx_global` is that `cs_ctx_alloc` allocates a new context structure each time it is called, while `cs_ctx_global` allocates a new context structure only once, the first time it is called. On subsequent calls, `cs_ctx_global` simply returns a pointer to the existing context structure.
- See also **`ct_con_alloc`**, **`ct_config`**, **`cs_ctx_drop`**, **`cs_ctx_global`**, **`cs_config`**

`cs_ctx_drop`

Description Deallocate a `CS_CONTEXT` structure.

Syntax `CS_RETCODE cs_ctx_drop(context)`

`CS_CONTEXT *context;`

Parameters *context*
A pointer to a `CS_CONTEXT` structure.

Return value **`cs_cxt_drop`** returns:

Returns	To Indicate
<code>CS_SUCCEED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.

`cs_ctx_drop` returns `CS_FAIL` if the context contains an open connection.

Examples

```
/*
** ex_ctx_cleanup()
**
```

```
** Parameters:
**   context   Pointer to context structure.
**   status    Status of last interaction with Client-
**             Library. If not ok, this routine will perform
**             a force exit.
**
** Returns:
**   Result of function calls from Client-Library.
*/
CS_RETCODE CS_PUBLIC
ex_ctx_cleanup(context, status)
CS_CONTEXT *context;
CS_RETCODE status;
{
    CS_RETCODE    retcode;
    CS_INT        exit_option;
    exit_option = (status != CS_SUCCEED) ? CS_FORCE_EXIT :
        CS_UNUSED;
    retcode = ct_exit(context, exit_option);
    if (retcode != CS_SUCCEED)
    {
        ex_error("ex_ctx_cleanup: ct_exit() failed");
        return retcode;
    }
    retcode = cs_ctx_drop(context);
    if (retcode != CS_SUCCEED)
    {
        ex_error("ex_ctx_cleanup: cs_ctx_drop() failed");
        return retcode;
    }
    return retcode;
}
```

Usage

- A CS_CONTEXT structure describes a particular context, or operating environment, for a set of server connections.
- Once a CS_CONTEXT has been deallocated, it cannot be used again. To allocate a new CS_CONTEXT, an application can call `cs_ctx_alloc`.
- A Client-Library application cannot call `cs_ctx_drop` to deallocate a CS_CONTEXT structure until it has called `ct_exit` to clean up Client-Library space associated with the context.

See also`cs_ctx_alloc`, `ct_close`, `ct_exit`

cs_ctx_global

Description Allocate or return a CS_CONTEXT structure.

Syntax CS_RETCODE cs_ctx_global(version, ctx_pointer)

```
CS_INT      version;
CS_CONTEXT  **ctx_pointer;
```

Parameters

version

One of the following symbolic values, to indicate the intended version of CS-Library behavior:

Value of <i>version</i>	To Indicate	Features Supported
CS_VERSION_100	10.0 behavior.	Initial version.
CS_VERSION_110	11.1 behavior.	Unicode character set support. Use of external configuration files to control Client-Library property settings.
CS_VERSION_120	12.0 behavior.	
CS_VERSION_125	12.5 behavior.	

If an application has already allocated a CS_CONTEXT structure, *version* must match the version previously requested.

ctx_pointer

The address of a pointer variable. `cs_ctx_global` sets **ctx_pointer* to the address of a new or previously allocated CS_CONTEXT structure.

In case of error, `cs_ctx_global` sets **ctx_pointer* to NULL.

Return value

`cs_ctx_global` returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_MEM_ERROR	The routine failed because it could not allocate sufficient memory.
CS_FAIL	The routine failed for other reasons.

Common reasons for a `cs_ctx_global` failure include:

- A lack of available memory
- A *version* value that does not match a previously requested version

The first `cs_ctx_global` call to execute in an application can fail due to configuration problems. See “Returns” under `cs_ctx_alloc` in this chapter for more information.

Usage

- **cs_ctx_alloc** also allocates a context structure. The only difference between **cs_ctx_alloc** and **cs_ctx_global** is that **cs_ctx_alloc** allocates a new context structure each time it is called, while **cs_ctx_global** allocates a new context structure only once, the first time it is called. On subsequent calls, **cs_ctx_global** simply returns a pointer to the existing context structure.
- **cs_ctx_global** is of use in applications that need to access a single context structure from multiple independent modules.
- For more information on context structures, see **cs_ctx_alloc** in this chapter.

See also

cs_ctx_alloc, cs_ctx_drop, cs_config, ct_con_alloc, ct_config

cs_diag

Description

Manage inline error handling.

Syntax

```
CS_RETCODE cs_diag(context, operation, type, index,  
                    buffer)
```

```
CS_CONTEXT *context;  
CS_INT      operation;  
CS_INT      type;  
CS_INT      index;  
CS_VOID     *buffer;
```

Parameters

context

A pointer to a CS_CONTEXT structure.

operation

The operation to perform. Table 2-4 on page 39 lists the legal symbolic values for *operation*.

type

Depending on the value of *operation*, *type* indicates either the type of structure to receive message information or the type of message on which to operate, or both.

Possible values are:

Value of <i>type</i>	To indicate
SQLCA_TYPE	A SQLCA structure.
SQLCODE_TYPE	A SQLCODE structure, which is a long integer.

Value of <i>type</i>	To indicate
SQLSTATE_TYPE	A SQLSTATE structure, which is a 6-element character array.
CS_CLIENTMSG_TYPE	A CS_CLIENTMSG structure. Also used to indicate CS-Library messages.

index

The index of the message of interest. The first message has an index of 1, the second an index of 2, and so forth.

buffer

A pointer to data space.

Depending on the value of *operation*, *buffer* can point to a structure or a CS_INT.

Return value

cs_diag returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_NOMSG	The application attempted to retrieve a message whose index is higher than the highest valid index. For example, the application attempted to retrieve message number 3 but only 2 messages were available.

Common reasons for a **cs_diag** failure include:

- Invalid *context*
- Inability to allocate memory
- Invalid parameter combination

Usage

Table 2-4: Summary of *cs_diag* parameter usage

Value of <i>operation</i>	<i>cs_diag</i>	<i>type</i> ls	<i>index</i> ls	<i>buffer</i> ls
CS_INIT	Initializes inline error handling.	CS_UNUSED	CS_UNUSE D	NULL
CS_MSGLIMIT	Sets the maximum number of messages to store.	CS_CLIENTMSG_TYPE	CS_UNUSE D	A pointer to an integer value.

Value of operation	cs_diag	type ls	index ls	buffer ls
CS_CLEAR	Clears message information for this context. If <i>buffer</i> is not NULL, cs_diag also clears the <i>*buffer</i> structure by initializing it with blanks and/or NULLs, as appropriate.	One of the legal <i>type</i> values.	CS_UNUSE D	A pointer to a structure whose type is defined by <i>type</i> , or NULL.
CS_GET	Retrieves a specific message.	One of the legal <i>type</i> values.	The one-based index of the message to retrieve.	A pointer to a structure whose type is defined by <i>type</i> .
CS_STATUS	Returns the current number of stored messages.	CS_CLIENTMSG_TYPE	CS_UNUSE D	A pointer to an integer value.

- An application that includes calls to CS-Library can handle CS-Library messages in one of two ways:
 - The application can call cs_config to install a CS-Library message callback, or
 - The application can handle CS-Library messages inline, using cs_diag.

An application can switch back and forth between the inline method and the callback method:

- Installing a CS-Library message callback turns off inline message handling. Any saved messages are discarded.
- Likewise, cs_diag(CS_INIT) “de-installs” an application’s CS-Library message callback. If the application has a message callback installed when cs_diag(CS_INIT) is called, the application’s first CS_GET call to cs_diag will retrieve a warning message to this effect.

If a CS-Library message callback is not installed and inline message handling is not enabled, CS-Library discards message information.

- `cs_diag` manages inline message handling for a specific context. If an application has more than one context, it must make separate `cs_diag` calls for each context.
- In a multithreaded application, `cs_diag` reports only those messages that pertain to CS-Library calls made by the thread which has called `cs_diag`. For more information on multithreaded applications, see the “Multithreaded Programming” topics page in the *Open Client Client-Library/C Reference Manual*.
- `cs_diag` allows an application to retrieve message information into a `CS_CLIENTMSG` structure or a `SQLCA`, `SQLCODE`, or `SQLSTATE` structure. When retrieving messages, `cs_diag` assumes that *buffer* points to a structure of the type indicated by *type*.

An application that is retrieving messages into a `SQLCA`, `SQLCODE`, or `SQLSTATE` structure must set the CS-Library context property `CS_EXTRA_INF` to `CS_TRUE`. This is because the SQL structures contain information that is not ordinarily returned by CS-Library’s error-handling mechanism.

An application that is not using the SQL structures can also set `CS_EXTRA_INF` to `CS_TRUE`. In this case, the extra information is returned as standard CS-Library messages.

- If `cs_diag` does not have sufficient internal storage space in which to save a new message, it throws away all unread messages and stops saving messages. The next time it is called with *operation* as `CS_GET`, it returns a special message to indicate the space problem.

After returning this message, `cs_diag` starts saving messages again.

Initializing inline error handling

- To initialize inline error handling, an application calls `cs_diag` with *operation* as `CS_INIT`.
- Generally, if a context will use inline error handling, the application should call `cs_diag` to initialize inline error handling for the context immediately after allocating it.

Clearing messages

- To clear message information for a context, an application calls `cs_diag` with *operation* as `CS_CLEAR`.
 - `cs_diag` assumes that *buffer* points to a structure whose datatype corresponds to the value of *type*.

- cs_diag clears the **buffer* structure by setting it to blanks and/or NULLs, as appropriate.
- Message information is not cleared until an application explicitly calls cs_diag with *operation* as CS_CLEAR. Retrieving a message does not remove it from the message queue.

Retrieving messages

- To retrieve message information, an application calls cs_diag with *operation* as CS_GET, *type* as the type of structure in which to retrieve the message, *index* as the one-based index of the message of interest, and **buffer* as a structure of the appropriate type.
- cs_diag fills in the **buffer* structure with the message information.
- If an application attempts to retrieve a message whose index is higher than the highest valid index, cs_diag returns CS_NOMSG to indicate that no message is available.
- See the “SQLCA Structure”, “SQLCODE Structure”, “SQLSTATE structure,” and “CS_CLIENTMSG Structure” topics pages in the *Open Client Client-Library/C Reference Manual* for information on these structures.

Limiting messages

- If an application runs on platforms with limited memory, you may want to limit the number of messages that CS-Library saves in that application.
- To limit the number of saved messages, an application calls cs_diag with *operation* as CS_MSGLIMIT and *type* as CS_CLIENTMSG_TYPE.
- When a message limit is reached, CS-Library discards any new messages.
- An application cannot set a message limit that is less than the number of messages currently saved.
- CS-Library’s default behavior is to save an unlimited number of messages. An application can restore this default behavior by setting a message limit of CS_NO_LIMIT.

Retrieving the number of messages

- To retrieve the number of current messages, an application calls cs_diag with *operation* as CS_STATUS and *type* as the CS_CLIENTMSG_TYPE.

See also

ct_callback, ct_options

cs_dt_crack

Description Convert a machine-readable datetime value into a user-accessible format.

Syntax CS_RETCODE cs_dt_crack(context, datatype, dateval,
daterec)

```
CS_CONTEXT    *context;
CS_INT        datatype;
CS_VOID       *dateval;
CS_DATEREC    *daterec;
```

Parameters

context

A pointer to a CS_CONTEXT structure.

datatype

A symbolic value indicating the datatype of **dateval*:

Value of <i>datatype</i>	To Indicate
CS_DATETIME_TYPE	A CS_DATETIME <i>*dateval</i> .
CS_DATETIME4_TYPE	A CS_DATETIME4 <i>*dateval</i> .

dateval

A pointer to a the datetime value to be converted.

daterec

A pointer to a CS_DATEREC structure. `cs_dt_crack` fills this structure with the translated datetime value. A CS_DATEREC is defined as follows:

```
typedef struct cs_daterec {
    CS_INT    dateyear;        /* year          */
    CS_INT    datemonth;      /* month         */
    CS_INT    datedmonth;    /* day of month  */
    CS_INT    datedyear;     /* day of year   */
    CS_INT    datedweek;     /* day of week   */
    CS_INT    datehour;      /* hour          */
    CS_INT    dateminute;    /* minute        */
    CS_INT    datesecound;   /* second        */
    CS_INT    datemsecond;   /* millisecond   */
    CS_INT    datetzone;     /* timezone      */
}
```

```
    } CS_DATERECD;
```

where:

dateyear is a value greater than or equal to 1900.

datemonth is a value from 0 to 11.

datedmonth is a value from 1 to 31.

datedyear is a value from 1 to 366.

datedweek is a value from 0 to 6.

datehour is a value from 0 to 23.

dateminute is a value from 0 to 59.

datesecond is a value from 0 to 59.

datemsecond is a value from 0 to 999.

datetzone is reserved for future use. *cs_dt_crack* does not set this field.

The meanings of these numbers vary according to an application's locale. For example, if localization information specifies that Sunday is the first day of the week, then a *datedweek* value of 2 represents Tuesday. If localization information specifies that Monday is the first day of the week, then a *datedweek* value of 2 represents Wednesday.

An application can call *cs_dt_info* to find out what date-related localization values are in effect.

The time zone field (*datetzone*) is reserved for future use. This field will not be set.

For more information on localization, see the "International Support" topics page in the *Open Client Client-Library/C Reference Manual*.

Return value

cs_dt_crack returns:

Returns	To Indicate
CS_SUCCEEDED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a *cs_dt_crack* failure is an invalid parameter.

Usage

- **cs_dt_crack** converts a datetime value into its integer components and places those components into a CS_DATERECD structure.

- Datetime values are stored in an internal format. For example, a CS_DATETIME value is stored as the number of days since January 1, 1900 plus the number of three hundredths of a second since midnight. `cs_dt_crack` converts a value of this type into a format that an application can more easily access.

See also `cs_dt_info`

cs_dt_info

Description Set or retrieve language-specific datetime information.

Syntax `CS_RETCODE cs_dt_info(context, action, locale, type, item, buffer, buflen, outlen)`

```
CS_CONTEXT *context;
CS_INT action;
CS_LOCALE *locale;
CS_INT type;
CS_INT item;
CS_VOID *buffer;
CS_INT buflen;
CS_INT *outlen;
```

Parameters *context*

A pointer to a CS_CONTEXT structure.

When retrieving datetime information, if *locale* is NULL, `cs_dt_info` uses the default locale information contained in this context structure.

action

One of the following symbolic values:

Value of <i>action</i>	<i>cs_dt_info</i>
CS_SET	Sets a datetime conversion format.
CS_GET	Retrieves datetime information.

locale

A pointer to a CS_LOCALE structure. A locale structure contains locale information, including datetime information.

When setting datetime information, *locale* must be supplied.

When retrieving datetime information, *locale* can be NULL. If *locale* is NULL, `cs_dt_info` uses the default locale information contained in **context*.

type

The type of information of interest. Table 2-5 lists the symbolic values that are legal for *type*.

item

When retrieving information, *item* is the item number of the type category to retrieve. For example, to retrieve the name of the first month, an application passes *type* as CS_MONTH and *item* as 0.

When setting a datetime conversion format, pass *item* as CS_UNUSED.

buffer

If datetime information is being retrieved, *buffer* points to the space in which cs_dt_info will place the requested information.

If *buflen* indicates that **buffer* is not large enough to hold the requested information, cs_dt_info sets **outlen* to the length of the requested information and returns CS_FAIL.

If a datetime conversion format is being set, *buffer* points to a symbolic value representing a conversion format.

buflen

The length, in bytes, of **buffer*.

If *item* is CS_12HOUR, pass *buflen* as CS_UNUSED.

outlen

A pointer to an integer variable.

cs_dt_info sets **outlen* to the length, in bytes, of the requested information.

If the requested information is larger than *buflen* bytes, an application can use the value of **outlen* to determine how many bytes are needed to hold the information.

Return value

cs_dt_info returns:

Returns	To Indicate
CS_SUCCEEDED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a cs_dt_info failure is an invalid parameter.

Usage

Table 2-5: Summary of cs_dt_info parameter usage

Value of <i>type</i>	<i>cs_dt_info</i>	<i>action</i> Can Be	<i>item</i> Can Be	<i>*buffer</i> Is
CS_MONTH	Retrieves the month name string.	CS_GET	0-11	A character string.

Value of <i>type</i>	<i>cs_dt_info</i>	action Can Be	<i>item Can Be</i>	*buffer Is
CS_SHORTMONTH	Retrieves the short month name string.	CS_GET	0–11	A character string.
CS_DAYNAME	Retrieves the day name string.	CS_GET	0–6	A character string.
CS_DATEORDER	Retrieves the date order string.	CS_GET	CS_UNUSED	A string containing the three characters “m,” “d,” and “y” to indicate the position of the month, day, and year in the datetime format.
CS_12HOUR	Retrieves whether or not the language uses 12-hour time formats.	CS_GET	CS_UNUSED	CS_TRUE if 12-hour formats are used; CS_FALSE if 24-hour formats are used.
CS_DT_CONVFMFMT	Sets or retrieves the datetime conversion format.	CS_GET or CS_SET	CS_UNUSED	A symbolic value. See the Comments section, below, for a list of possible values.

- `cs_dt_info` sets or retrieves native language-specific datetime information:
 - `cs_dt_info` can return native language date part names, date part ordering information, datetime format information, and whether or not the language uses 12-hour date formats.
 - `cs_dt_info` can set datetime format information.
- If *locale* is NULL, `cs_dt_info` looks for native language locale information in **context*. An application can set locale information for a CS_CONTEXT by calling `cs_config` with *property* as CS_LOC_PROP.

If not specifically set, locale information in a CS_CONTEXT defaults to information that CS-Library picks up from the operating system when the context is allocated. If locale information is not available from the operating system, CS-Library uses platform-specific localization values in the new context.

- A locale’s date-order string, which can be retrieved by calling `cs_dt_info` with *type* as CS_DATEORDER, describes how ambiguous date strings are resolved when converting from character datatypes to CS_DATETIME or CS_DATETIME4. For example, “04/05/96” could be interpreted as “April 5, 1996” or “May 4, 1996”. The former result corresponds to the date-order string of “mdy”, and the latter corresponds to “dmy”.

Although an application cannot set a locale’s date-order string directly, it can call `cs_locale` and change the national-language used when converting dates. To do this, the application calls `cs_locale` with *action* as `CS_SET`, *type* as `CS_LC_TIME`, and **buffer* as a locale name. The application can specify a locale whose national language is configured to use a different date-order string. A national language’s date-order string is configured as follows:

- For each national language, a *common.loc* file is located in a language subdirectory in the `/$SYBASE/locales/messages` subdirectory.
- The “dateformat” setting in the [datetime] section of the file specifies the date-order string. For example:

```
[datetime]
    dateformat=dmy
```

For more information on the *common.loc* file, see the *Open Client/Server Configuration Guide*.

- The date conversion format, which can be set or retrieved by calling `cs_dt_info` with *type* as `CS_DT_CONVFM`, describes the format of the result when a `CS_DATETIME` or `CS_DATETIME4` value is converted to a character-based datatype.
- Table 2-6 lists the values that are legal for **buffer* when *type* is `CS_DT_CONVFM`:

Table 2-6: Values for *buffer when type is CS_DT_CONVFM (cs_dt_info)

Symbolic Value	To Indicate
CS_DATES_HMS	hh:mm:ss 16:15:31
CS_DATES_SHORT	monthname dd yyyy hh:mm [am pm] October 19 1961 04:15:31 pm
CS_DATES_LONG	monthname dd yyyy hh:mm:ss:zzz [am pm] October 19 1961 04:15:31:665 pm
CS_DATES_MDY1	mm/dd/yy 10/19/61
CS_DATES_MYD1	mm/yy/dd 10/61/19
CS_DATES_DMY1	dd/mm/yy 19/10/61

Symbolic Value	To Indicate
CS_DATES_DYM1	dd/yy/mm 19/61/10
CS_DATES_YDM1	yy/dd/mm 61/19/10
CS_DATES_YMD2	yy/mm/dd 61/10/19
CS_DATES_MDY1_YYYY	mm/dd/yyyy 10/19/1961
CS_DATES_DMY1_YYYY	dd/mm/yyyy 19/10/1961
CS_DATES_YMD2_YYYY	yyyy/mm/dd 1961/10/19
CS_DATES_DMY2	dd.mm.yy 19.10.61
CS_DATES_YMD1	yy.mm.dd 61.10.19
CS_DATES_DMY2_YYYY	dd.mm.yyyy 19.10.1961
CS_DATES_YMD1_YYYY	yyyy.mm.dd 1961.10.19
CS_DATES_DMY4	dd monthname yy 19 January 61
CS_DATES_DMY4_YYYY	dd monthname yyyy 19 January 1961
CS_DATES_MDY2	monthname dd, yy January 19, 61
CS_DATES_MDY2_YYYY	monthname dd, yyyy January 19, 1961
CS_DATES_DMY3	dd-mm-yy 19-10-61
CS_DATES_MDY3	mm-dd-yy 10-19-61

Symbolic Value	To Indicate
CS_DATES_DMY3_YYYY	dd-mm-yyyy 19-10-1961
CS_DATES_MDY3_YYYY	mm-dd-yyyy 10-19-1961
CS_DATES_YMD3	yymmdd 611019
CS_DATES_YMD3_YYYY	yyymmdd 19611019

- A `cs_locale(CS_SET, CS_LC_TIME)` call or a `cs_locale(CS_SET, CS_LC_ALL)` call resets date/time conversion information to the default settings for the specified national language.

See also `cs_dt_crack`, `cs_locale`

cs_loc_alloc

Description Allocate a CS_LOCALE structure.

Syntax CS_RETCODE cs_loc_alloc(context, loc_pointer)

CS_CONTENT *context;
CS_LOCALE **loc_pointer;

Parameters *context*
A pointer to a CS_CONTEXT structure.

loc_pointer
The address of a pointer variable. `cs_loc_alloc` sets *loc_pointer* to the address of a newly allocated CS_LOCALE structure.

Return value **cs_loc_alloc** returns:

Returns	To Indicate
CS_SUCCEEDED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a `cs_loc_alloc` failure is a lack of adequate memory.

- Usage
- An Open Client/Server application can use a `CS_LOCALE` structure to define custom localization values for a context, thread, connection, or data element. To define custom localization values, an application:
 - Calls `cs_loc_alloc` to allocate a `CS_LOCALE` structure.
 - Calls `cs_locale(CS_SET)` to load the `CS_LOCALE` with custom values.
 - Uses the `CS_LOCALE` to set the `CS_LOC_PROP` property for a context or connection; calls `srv_thread_props` to set the `SRV_T_LOCALE` property for a thread; uses the `CS_LOCALE` in a `CS_DATAFMT` structure that describes a program variable; or uses the `CS_LOCALE` as a parameter to an Open Client/Server routine.
 - Calls `cs_loc_drop` to drop the `CS_LOCALE`.
 - Localization values define:
 - The language and character set to use for Open Client/Server and Adaptive Server messages
 - How to represent dates and times
 - The character set to use when converting data to and from character datatypes
 - The collating sequence used to define the sort order used by `cs_strcmp`
- See also `cs_ctx_alloc`, `cs_loc_drop`, `cs_locale`

cs_loc_drop

Description Deallocate a `CS_LOCALE` structure.

Syntax `CS_RETCODE cs_loc_drop(context, locale)`

```
CS_CONTEXT    *context;
CS_LOCALE     *locale;
```

Parameters

context

A pointer to the `CS_CONTEXT` structure that represents the context in which the `CS_LOCALE` was allocated.

locale

A pointer to a `CS_LOCALE` structure.

Return value **cs_loc_drop** returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

- Usage
- A CS_LOCALE structure contains localization information.
 - Once a CS_LOCALE structure has been deallocated, it cannot be used again. To allocate a new CS_LOCALE structure, an application can call cs_loc_alloc.
 - An application should take care to ensure that it does not deallocate a CS_LOCALE structure that is still in use. A CS_LOCALE structure is considered to be in use if a CS_DATAFMT structure references it.
 - An application can deallocate a CS_LOCALE structure after calling cs_config or ct_con_props to set the CS_LOC_PROP property for a context or connection. This is because cs_config and ct_con_props copy information from the user-supplied CS_LOCALE structure rather than setting up direct references to it.

See also cs_loc_alloc, cs_locale

cs_locale

Description Load a CS_LOCALE structure with localization values or retrieve the locale name previously used to load a CS_LOCALE structure.

Syntax CS_RETCODE cs_locale(context, action, locale, type, buffer, buflen, outlen)

```
CS_CONTEXT *context;
CS_INT action;
CS_LOCALE *locale;
CS_INT type;
CS_CHAR *buffer;
CS_INT buflen;
CS_INT *outlen;
```

Parameters *context*
A pointer to the CS_CONTEXT structure that represents the context in which the CS_LOCALE was allocated.

action

One of the following symbolic values:

Value of <i>action</i>	cs_locale
CS_SET	Loads the CS_LOCALE with new localization values.
CS_GET	Retrieves the locale name that was used to load the CS_LOCALE.

locale

A pointer to a CS_LOCALE structure. If *action* is CS_SET, *cs_locale* modifies this structure. If *action* is CS_GET, *cs_locale* examines the structure to determine the locale name that was previously used to load it.

type

One of the following symbolic values:

Value of <i>type</i>	To Indicate
CS_LC_ALL	All types of localization information. Note CS_LC_ALL is “set only”; that is, <i>action</i> must be CS_SET when <i>type</i> is CS_LC_ALL.
CS_LC_COLLATE	The collating sequence (also called “sort order”). Open Client uses a collating sequence when sorting and comparing character data.
CS_LC_CTYPE	The character set. Open Client uses a character set when it converts to or from character datatypes.
CS_LC_MESSAGE	The language and character set to use for Open Client/Server and Adaptive Server error messages.
CS_LC_TIME	The language and character set to use when converting between datetime and character datatypes. CS_LC_TIME controls month names and abbreviations, datepart ordering, and whether the “am/pm” string is used.
CS_SYB_LANG, CS_SYB_CHARSET, CS_SYB_SORTORDER, CS_SYB_LANG_CHARSET	For information on these values, see “Using language, character set, and sort order names with <i>cs_locale</i> ” on page 56.

Warning! Open Server application programmers must set *type* to CS_LC_ALL when configuring the CS_LOCALE structure that applies to the Open Server application as a whole.

buffer

If *action* is CS_SET, *buffer* points to a character string that represents a locale name, a character set name, a language name, a sort order name, or a language/character set pair.

If *action* is CS_GET, *buffer* points to the space in which cs_locale will place a locale name, a character set name, a language name, a sort order name, or a language/character set pair. On output, all names are null-terminated. The buffer must be long enough for the name plus a null terminator.

buflen

The length, in bytes, of **buffer*.

If *action* is CS_SET and the value in **buffer* is null-terminated, pass *buflen* as CS_NULLTERM.

outlen

A pointer to an integer variable.

outlen is not used if *action* is CS_SET.

If *action* is CS_GET and *outlen* is supplied, cs_locale sets **outlen* to the length, in bytes, of the locale name.

If the name is larger than *buflen* bytes, an application can use the value of **outlen* to determine how many bytes are needed to hold the name.

If *action* is CS_SET or if an application does not require return length information, it can pass *outlen* as NULL.

Return value

cs_locale returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a cs_locale failure include:

- *action* is CS_SET and the **buffer* locale name cannot be found in the Sybase locales file.
- *action* is CS_GET and *buflen* indicates that the **buffer* data space is too small.
- Missing localization files.

Usage

Note `cs_locale`'s behavior depends on platform-specific configuration issues. You must read the localization chapter in the *Open Client/Server Configuration Guide* to obtain a full understanding of Client-Library's localization mechanism. For a discussion of programming issues related to localization, see the *Open Client/Server International Developer's Guide*.

- `cs_locale(CS_SET)` loads a `CS_LOCALE` structure with localization values. `cs_locale(CS_GET)` retrieves current settings from the `CS_LOCALE` structure.
- A *locale name* is a character string that represents a language/ character set/sort order combination. For example, the locale name "fr" might represent the language/character set/sort order combination "French, iso_1, binary."
 - Sybase predefines some locale names in the default locales file.
 - A System Administrator can define additional locale names and add them to the Sybase locales file. The *Open Client/Server Configuration Guide* contains instructions for adding locale names.
- For more information on localization, see the *Open Client/Server International Developer's Guide*.

Loading a CS_LOCALE structure

- An application needs to initialize, or "load", a `CS_LOCALE` before using it to define custom localization values for a context, connection, or data element.
- `cs_locale(CS_SET)` loads a `CS_LOCALE` structure with localization values. Any localization value can be specified by giving a locale name. Character sets, languages, and sort orders can also be specified directly by name.
- When specifying a locale name, *buffer* must specify a name that corresponds to an entry in the Sybase locales file.

buffer can also be passed as `NULL` to specify the default locale. In this case, `cs_locale` searches the operating system for a locale name to use. If an appropriate locale name cannot be found in the operating system environment, `cs_locale` uses a platform-dependent default locale name.

The localization item(s) of interest are loaded based on the configuration of the locales file entry. For more information about the locales file and the `cs_locale` search process, see the *Open Client/Server Configuration Guide*.

- For instructions for directly specifying character set, language, or sort order names, see “Using language, character set, and sort order names with cs_locale” on page 56.
- After loading a CS_LOCALE with custom values, an application can:
 - Call cs_config with *property* as CS_LOC_PROP to copy the custom localization values into a context structure.
 - Call ct_con_props with *property* as CS_LOC_PROP to copy the custom localization values into a connection structure.
 - Supply the CS_LOCALE as a parameter to a routine that accepts custom localization values (cs_dt_info, cs_strcmp, cs_time).
 - Include the CS_LOCALE in a CS_DATAFMT structure describing a destination program variable (cs_convert, ct_bind).
- Because cs_config copies locale information, an application can deallocate a CS_LOCALE structure after calling cs_config to set the CS_LOC_PROP property. Likewise, an application can deallocate a CS_LOCALE structure after calling ct_con_props to set the CS_LOC_PROP property. If a CS_DATAFMT structure uses a CS_LOCALE structure, however, the application must not deallocate the CS_LOCALE until the CS_DATAFMT no longer references it.
- The first time a locale name is referenced, all localization information for the language, character set, and sort order that the locale name identifies is read from the environment and cached into **context*. If this locale name is referenced again, cs_locale will read the information from the CS_CONTEXT instead of the environment.

Retrieving a Locale name

- An application can retrieve the locale name that was used to load a CS_LOCALE by calling cs_locale(CS_GET) with *type* as the type of localization information of interest and *locale* as a pointer to the CS_LOCALE structure.
- cs_locale sets **buffer* to a null-terminated character string representing the locale name that was used to load the CS_LOCALE.

Using language, character set, and sort order names with cs_locale

- It is possible for an application to use language, character set, and sort order names, instead of a locale name, when calling cs_locale.

- To use a language, character set, or sort order name, an application calls `cs_locale` with *type* as `CS_SYB_LANG`, `CS_SYB_CHARSET`, `CS_SYB_SORTORDER`, or `CS_SYB_LANG_CHARSET`. The following table summarizes `cs_locale`'s parameters for these values of *type*:

Table 2-7: Using language, character set, and sort order names with `cs_locale`

Value of <i>type</i>	action ls	buffer ls	<i>cs_locale</i>
CS_SYB_LANG	CS_SET	A pointer to a language name.	Loads the CS_LOCALE with the specified language information.
	CS_GET	A pointer to data space.	Places the current language name in <i>*buffer</i> . The name is null terminated.
CS_SYB_CHARSET	CS_SET	A pointer to a character set name.	Loads the CS_LOCALE with the specified character set information.
	CS_GET	A pointer to data space.	Places the current character set name in <i>*buffer</i> . The name is null terminated.
CS_SYB_SORTORDER	CS_SET	A pointer to a sort order name.	Loads the CS_LOCALE with the specified sort order information.
	CS_GET	A pointer to data space.	Places the current sort order name in <i>*buffer</i> . The name is null terminated.
CS_SYB_LANG_CHARSET	CS_SET	A pointer to a string of the form <i>language_name.character_set_name</i> .	Loads the CS_LOCALE with the specified language and character set information.
	CS_GET	A pointer to data space.	Places a string of the form <i>language_name.character_set_name</i> in <i>*buffer</i> . The string is null terminated.

- The application must have previously loaded the CS_LOCALE structure with consistent information by calling `cs_locale` with *type* as `CS_LC_ALL`.
- If an application specifies only a language name, then `cs_locale` uses the character set and sort order already specified in the preloaded CS_LOCALE structure.

If an application specifies only a character-set name, then `cs_locale` uses the language and sort order already specified in the preloaded CS_LOCALE structure.

If an application specifies only a sort-order name, then `cs_locale` uses the language and character set already specified in the preloaded CS_LOCALE structure.

If a language, character set, and sort-order combination is not valid, `cs_locale` returns `CS_FAIL`.

- Valid language names correspond to subdirectories in the `$$SYBASE/locales` directory. Valid character-set names correspond to subdirectories in the `$$SYBASE/charsets` directory. Valid sort-order names for a character set correspond to file names, stripped of any suffix, in the `$$SYBASE/charsets/character_set_name` directory.
- If the required localization files for the requested language or character set do not exist, `cs_locale` returns `CS_FAIL`.

See also `cs_loc_alloc`, `cs_loc_drop`

cs_manage_convert

Description Install or retrieve a user-defined character-set conversion routine.

Syntax `CS_RETCODE cs_manage_convert(context, action, srctype, srcname, srcnamelen, desttype, destname, destnamelen, conv_multiplier, func)`

```
CS_CONTEXT *context;
CS_INT action;
CS_INT srctype;
CS_CHAR *srcname;
CS_INT srcnamelen;
CS_INT desttype;
CS_CHAR *destname;
CS_INT destnamelen;
CS_INT *conv_multiplier;
CS_CONV_FUNC *func;
```

Parameters *context*
A pointer to a `CS_CONTEXT` structure.

action
One of the following symbolic values:

Value of <i>action</i>	<i>cs_manage_convert</i>
<code>CS_SET</code>	Installs a conversion routine and conversion multiplier for conversions between the indicated datatypes and character-set names.
<code>CS_GET</code>	Retrieves the current conversion routine and conversion multiplier for the indicated datatypes and character-set names.

Value of <i>action</i>	<i>cs_manage_convert</i>
CS_CLEAR	Clears the current conversion routine by replacing it with CS-Library's default conversion routine for the indicated datatypes and character-set names.

srctype

The datatype of the source data for the conversion. In the current version, *srctype* must be CS_CHAR_TYPE.

srcname

The name of the character set associated with *srctype*. This name must correspond to the name of a subdirectory within the *charsets* subdirectory of the Sybase installation directory.

srcnamelen

The length, in bytes, of *srcname*. If *srcname* is null-terminated, *srcnamelen* can be passed as CS_NULLTERM.

desttype

The datatype of the destination data. In the current version, *desttype* must be CS_CHAR_TYPE.

destname

The name of the destination character set. This name must correspond to the name of a subdirectory within the *charsets* subdirectory of the Sybase installation directory.

destnamelen

The length, in bytes, of *destname*. If *destname* is null-terminated, *destnamelen* can be passed as CS_NULLTERM.

conv_multiplier

The address of a CS_INT variable. When action is CS_SET, pass **conv_multiplier* as the conversion multiplier for the indicated character-set conversion. When action is CS_GET, **conv_multiplier* receives the conversion multiplier for the indicated character-set conversion. When action is CS_CLEAR, pass *conv_multiplier* as NULL.

See "Meaning of the conversion multiplier" on page 62 for a explanation of how applications use this number.

func

The address of a CS_CONV_FUNC variable, which itself is a pointer to a character-set conversion routine. “Defining a custom character set conversion routine” on page 62 describes the requirements for coding a custom character-set conversion routine.

If a conversion routine is being installed, **func* points to the conversion routine to be installed.

If a conversion routine is being retrieved, cs_manage_convert sets **func* to point to the currently installed character-set conversion routine for *srcname* to *destname* conversions, or to NULL if no custom routine is installed.

If a conversion routine is being cleared, pass **func* as NULL.

Note *func* represents a pointer to a pointer to a function. There are special requirements for passing this parameter. See the example code fragment under “Installing a custom character set conversion routine” on page 64.

Return value

cs_manage_convert returns:

Returns	To Indicate
CS_SUCCEEDED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a cs_manage_convert failure is an invalid parameter.

Usage

- **cs_manage_convert** allows an application to install a custom character-set conversion routine that converts data from one character set to another.

Character set conversion

- Client-Library, CS-Library, and Server-Library can all perform character-set conversion. Character-set conversion occurs when an application converts between any two character datatypes and associates different character sets with the source and destination.
 - In CS-Library, cs_convert performs character-set conversion when converting between two character datatypes if the *destfmt* CS_DATAFMT structure specifies (or defaults to) a different locale than the *srcfmt* CS_DATAFMT structure.

- In Client-Library, an application can request character-set conversion for fetched character data by binding the column to a character-datatype variable and passing a pointer to a `CS_LOCALE` in `ct_bind`'s `datafmt` that is different from the connection's locale (that is, the `CS_LOC_PROP` connection property).
- In Server-Library, all character data sent to a client or received from a client is automatically converted between the client thread's character set and the Open Server character set.
- The character datatypes are `CS_CHAR`, `CS_LONGCHAR`, `CS_TEXT`, `CS_UNICHAR` and `CS_VARCHAR`.
- `cs_manage_convert` requires an application to pass both `srctype` and `desttype` as `CS_CHAR_TYPE`. However, CS-Library, Client-Library, and Server-Library will call the conversion routine to convert between any two character-based types when the conversion locales specify the character sets associated with the conversion routine.
- The most common reason for installing a custom conversion routine is to improve performance by replacing an indirect conversion with a direct conversion.

A custom character-set conversion routine can improve performance in applications that rely on character-set conversions where CS-Library does not use direct character-set conversion. Indirect character-set conversion converts first to Unicode UTF-8, and then from Unicode UTF-8 to the destination character set. Applications that perform these conversions can improve performance by installing a custom routine that supports direct conversion.

For example, an Open Server application could install a custom routine to convert between ISO 8859-1 and EUC JIS. This direct conversion may be faster than the indirect conversion (ISO 8859-1 to/from Unicode UTF-8 to/from EUC JIS) that is supplied with Open Server.

- To find out whether a specific character conversion is direct or indirect, look in the source character set's conversion configuration file. If there is an entry for the destination character set, then the conversion is direct. Character set configuration files are described in the *Open Client/Server International Developer's Guide*.
- For more information on character-set conversion, see the *Open Client/Server International Developer's Guide*.

Meaning of the conversion multiplier •

- Applications must provide `cs_manage_convert` with a conversion multiplier for conversions between the indicated character sets.
- The value of the conversion multiplier equals the largest number of bytes in the destination result that can replace one source byte when converting between the indicated character sets.
 - Applications can retrieve the conversion multiplier for a specific character-set conversion with `cs_conv_mult`. This number allows the application to determine the destination space needed for a conversion.

Defining a custom character set conversion routine •

A custom character-set conversion routine is defined as follows:

```
CS_RETCODE CS_PUBLIC
convfunc(context, srcfmt, srcdata,
         destfmt, destdata, destlen)
CS_CONTEXT      *context;
CS_DATAFMT     *srcfmt;
CS_VOID        *srcdata;
CS_DATAFMT     *destfmt;
CS_VOID        *destdata;
CS_INT         *destlen;
```

where:

context is a pointer to a `CS_CONTEXT` structure.

srcfmt is a pointer to a `CS_DATAFMT` structure describing the source data. *srcfmt*→*maxlength* describes the actual length, in bytes, of the source data.

srcdata is a pointer to the source data.

destfmt is a pointer to a `CS_DATAFMT` structure describing the destination data. *destfmt*→*maxlength* describes the actual length, in bytes, of the destination data space.

destdata is a pointer to the destination data space.

destlen is a pointer to an integer. The conversion routine should set **destlen* to the number of bytes placed in **destdata*. If the routine writes a truncated result, it should set **destlen* as the number of bytes written before truncation.

Note When converting into a CS_VARCHAR structure, the conversion routine should set both **destlen* and the CS_VARCHAR's *len* field to the number of bytes written to the CS_VARCHAR's *str* field.

- *cs_config* is the only CS-Library, Client-Library, or Server-Library function that can be called from within a custom conversion routine.
- A custom character-set conversion routine can return any of the values listed in Table 2-8.
 - If the conversion routine returns a value from Table 2-8 other than CS_SUCCEED, then the application receives a Client-Library or CS-Library message that corresponds to the indicated error condition.
 - If the conversion routine returns a value that is not listed in Table 2-8, then the application receives an “Unknown return code” error message from Client-Library or CS-Library.

Table 2-8: Return values for a custom conversion routine

Return Value	To Indicate
CS_SUCCEED	Successful conversion.
CS_TRUNCATED	The conversion resulted in truncation.
CS_MEM_ERROR	A memory allocation failure has occurred.
CS_EBADXLT	Some characters could not be translated.
CS_ENOXLT	The requested translation is not supported.
CS_EDOMAIN	The source value is outside the domain of legal values for the datatype.
CS_EDIVZERO	Division by 0 is not allowed.
CS_EOVERFLOW	The conversion resulted in overflow.
CS_EUNDERFLOW	The conversion resulted in underflow.
CS_EPRECISION	The conversion resulted in loss of precision.
CS_ESCALE	An illegal scale value was encountered.
CS_ESYNTAX	The conversion resulted in a value which is not syntactically correct for the destination type.
CS_ESTYLE	The conversion operation was stopped due to a style error.

Installing a custom character set conversion routine

- The following code demonstrates calling `cs_manage_convert` to install a custom conversion routine. The code is based on the assumption that the installed routine has been defined correctly. (See “Defining a custom character set conversion routine” on page 62.) The program variable `p_conv_func` is used to pass the address of the conversion routine.

```
#define MULT_ISO_1_TO_EUCJIS 4
CS_CONV_FUNC p_conv_func;
CS_INT      conv_mult = MULT_ISO_1_TO_EUCJIS;

/*
** Install the routine charconv_iso_1_TO_eucjis() to convert
** character data from iso_1 character set to eucjis character
** set.
*/
p_conv_func = charconv_iso_1_TO_eucjis;
if (cs_manage_convert(context, CS_SET,
    CS_CHAR_TYPE, "iso_1", CS_NULLTERM,
    CS_CHAR_TYPE, "eucjis", CS_NULLTERM,
    &conv_mult, &p_conv_func )
    != CS_SUCCEEDED)
{
    fprintf(stdout, "cs_manage_convert() failed!\n");
    (CS_VOID)ct_exit(context, CS_FORCE_EXIT);
    (CS_VOID)cs_ctx_drop(context);
    exit(-1);
}
```

See also `cs_conv_mult`, `cs_convert`, `cs_locale`, `cs_set_convert`

cs_objects

Description Save, retrieve, or clear objects and data associated with them.

Syntax `CS_RETCODE cs_objects(context, action, objname, objdata)`

```
CS_CONTEXT *context;
CS_INT      action;
CS_OBJNAME *objname;
CS_OBJDATA *objdata;
```

Parameters *context*
A pointer to a `CS_CONTEXT` structure.

action

One of the following symbolic values:

Value of <i>action</i>	<i>cs_objects</i>
CS_SET	Saves an object.
CS_GET	Retrieves the first matching object that it finds.
CS_CLEAR	Clears all matching objects.

objname

A pointer to an object name structure. **objname* names and describes the object of interest. An object name structure is defined as follows:

```

/*
** CS_OBJNAME
*/
typedef struct _cs_objname
{
    CS_BOOL        thinkexists;
    CS_INT         object_type;
    CS_CHAR        last_name[CS_MAX_NAME];
    CS_INT         lnlen;
    CS_CHAR        first_name[CS_MAX_NAME];
    CS_INT         fnlen;
    CS_VOID        *scope;
    CS_INT         scopelen;
    CS_VOID        *thread;
    CS_INT         threadlen;
} CS_OBJNAME;

```

The *object_type*, *last_name*, *first_name*, *scope*, and *thread* fields form a 5-part key that identifies a stored object (see “cs_objects naming keys” on page 69 for more information). The following table describes the CS_OBJNAME fields:

Table 2-9: CS_OBJNAME fields

Field	Description	Notes
<i>thinkexists</i>	Indicates whether the application expects this object to exist.	The value of <i>thinkexists</i> affects the <i>cs_objects</i> return code. For more information, see the Return values.

Field	Description	Notes
<i>object_type</i>	The type of the object.	This field is the first part of a 5-part key. <i>object_type</i> can be one of these values: <ul style="list-style-type: none"> • CS_CONNECTNAME • CS_CURSORNAME • CS_STATEMENTNAME • CS_CURRENT_CONNECTION • CS_WILDCARD (matches any value) • A user-defined value. User-defined values must be ≥ 100.
<i>last_name</i>	The “last name” associated with the object of interest, if any.	This field is the second part of a 5-part key.
<i>lnlen</i>	The length, in bytes, of <i>last_name</i> .	Can be CS_NULLTERM to indicate a null-terminated <i>last_name</i> . Can be CS_UNUSED to indicate an internal “unused” value for <i>last_name</i> . For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match any <i>last_name</i> value.
<i>first_name</i>	The “first name” associated with the object of interest, if any.	This field is the third part of a 5-part key.
<i>fnlen</i>	The length, in bytes, of <i>first_name</i> .	Can be CS_NULLTERM to indicate a null-terminated <i>first_name</i> . Can be CS_UNUSED to indicate an internal “unused” value for <i>first_name</i> . For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match any <i>first_name</i> value.
<i>scope</i>	Data that describes the scope of the object.	This field is the fourth part of a 5-part key.
<i>scopelen</i>	The length, in bytes, of <i>scope</i> .	Can be CS_NULLTERM to indicate null-terminated scope data. Can be CS_UNUSED to indicate an internal “unused” value for <i>*scope</i> . For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match any <i>scope</i> value.

Field	Description	Notes
<i>thread</i>	Platform-specific data that is used to distinguish threads in a multi-threaded execution environment.	This field is the fifth part of a 5-part key.
<i>threadlen</i>	The length, in bytes, of <i>thread</i> .	Can be CS_NULLTERM to indicate null-terminated thread data. Can be CS_UNUSED to indicate an internal “unused” value for <i>*thread</i> . For CS_GET and CS_CLEAR operations, can be CS_WILDCARD to match any <i>thread</i> value.

objdata

A pointer to an object data structure. **objdata* is the object of interest and any data associated with it. An object data structure is defined as follows:

```
/*
** CS_OBJDATA
*/
typedef struct _cs_objdata
{
    CS_BOOL          actuallyexists;
    CS_CONNECTION    *connection;
    CS_COMMAND       *command;
    CS_VOID          *buffer;
    CS_INT           buflen;
} CS_NAMEDATA;
```

The following table describes the CS_OBJDATA fields:

Table 2-10: CS_OBJDATA fields

Field	Description	Notes
<i>actuallyexists</i>	Indicates whether this object actually exists.	cs_objects sets <i>actuallyexists</i> to CS_TRUE if it finds a matching object. cs_objects sets <i>actuallyexists</i> to CS_FALSE if it does not find a matching object.
<i>connection</i>	A pointer to the CS_CONNECTION structure representing the connection in which the object exists.	

Field	Description	Notes
<i>command</i>	A pointer to the CS_COMMAND structure representing the command space with which the object is associated.	Can be NULL.
<i>buffer</i>	A pointer to data space. An application can use <i>buffer</i> to associate data with a saved object.	If <i>action</i> is CS_SET, <i>buffer</i> contains the data to associate with the object. If <i>action</i> is CS_GET, cs_objects sets <i>buffer</i> to the data associated with the object being retrieved.
<i>buflen</i>	The length, in bytes, of <i>buffer</i> .	If <i>action</i> is CS_SET, <i>buflen</i> is the length of the data contained in <i>buffer</i> . Can be CS_NULLTERM to indicate null-terminated data. Can be CS_UNUSED to indicate that there is no data associated with the object being saved. If <i>action</i> is CS_GET, <i>buflen</i> is the maximum capacity of <i>buffer</i> . cs_objects overwrites <i>buflen</i> with the number of bytes copied to <i>buffer</i> . If <i>buflen</i> is CS_UNUSED, cs_objects overwrites <i>buflen</i> with the length of the data but does not copy it to <i>buffer</i> .

Return value

cs_objects returns CS_SUCCEED or CS_FAIL depending on the values passed as *action* and *objname*→*thinkexists* (See Table 2-9 on page 65). The following table lists the return code for each combination:

Table 2-11: cs_objects return values

cs_objects action As	Called With		cs_objects Returns		
	<i>objname</i> → <i>thinkexists</i> As		No Match	Last-Name Match	Full Match
CS_GET	CS_TRUE		CS_FAIL	CS_FAIL	CS_SUCCEED
CS_GET	CS_FALSE		CS_SUCCEED	CS_SUCCEED	CS_SUCCEED
CS_SET	CS_TRUE		CS_FAIL	CS_FAIL	CS_SUCCEED
CS_SET	CS_FALSE		CS_SUCCEED	CS_SUCCEED	CS_FAIL
CS_CLEAR	CS_TRUE		CS_FAIL	CS_FAIL	CS_SUCCEED
CS_CLEAR	CS_FALSE		CS_SUCCEED	CS_SUCCEED	CS_SUCCEED

Usage

Table 2-12: Summary of *cs_objects* parameter usage

Value	<i>ofaction</i>	<i>objname is</i>	<i>objdata is</i>
CS_SET		A five-part key for the object.	The object to save and any additional data to save with it.
CS_GET		A five-part key for the object.	Set to the retrieved object.
CS_CLEAR		A five-part key for the object.	CS_UNUSED

- *cs_objects* is useful in precompiler applications that need to retrieve structures and data items by name.

***cs_objects* naming keys**

- *cs_objects* uses a five-part key, composed of the *object_type*, *last_name*, *first_name*, *scope*, and *thread* fields of **objname* structure.
 - On CS_SET operations, *cs_objects* uses this key to store the **objdata* object.
 - On CS_GET operations, *cs_objects* uses this key to retrieve an object specification into **objdata*.
 - On CS_CLEAR operations, *cs_objects* clears all objects that match the key.
- The following table describes the rules that *cs_objects* uses to determine whether or not key fields match:

Table 2-13: *cs_objects* key matching rules

<i>*objname</i> key length is	Stored key length is CS_UNUSED	Stored key length is another legal value
CS_WILDCARD	Match	Match
CS_UNUSED	Match	No match
Another Legal Value	No match	Match, if the names match and have the same length.

- *cs_objects* can achieve two types of matches:
 - “last-name matches,” in which the *last_name*, *scope*, and *thread* parts of the key match.
 - “full matches,” in which all five parts of the key match.

The type of match that *cs_objects* achieves, together with *action* and *objname*→*thinkexists*, determine its return code.

- On CS_GET and CS_CLEAR operations, an application may specify CS_WILDCARD for one or more **objname* key fields:
 - On a CS_GET operation, cs_objects sets **objdata* to reflect the first matching object that it finds.
 - On a CS_CLEAR operation, cs_objects clears all matching objects.

Retrieving “Current Connection” objects

- If an application has previously saved a CS_CURRENT_CONNECTION object, it can retrieve the current connection by:
 - Calling cs_objects with *objname*→*object_type* as CS_CURRENT_CONNECTION, *lnlen* as CS_UNUSED, and *flen* as CS_UNUSED. cs_objects ignores the *last_name* and *first_name* fields of *objname*, and sets *objdata*→*buffer* to the name of the current connection and *objdata*→*buflen* to the length of this name.
 - Calling cs_objects with *objname*→*object_type* as CS_CONNECTNAME and *objname*→*last_name* and *objname*→*lnlen* as the newly retrieved connection name and name length. cs_objects sets *objdata* to the retrieved connection.

Warning! An application cannot call cs_objects(CS_SET) from within a completion callback routine.

See also

cs_ctx_alloc

cs_prop_ssl_localid

Description Used to specify the path to the local ID (certificates) file.

Syntax

```
typedef struct _cs_sslid
{
    CS_CHAR    *identity_file;
    CS_CHAR    *identity_password;
} CS_SSLIDENTITY
```


Parameters

identity_file
 provides a path to the file containing a digital certificate and the associated private key.

CS_GET only returns the *identity_file* used, and only if it is set with CS_CONNECTION.

identity_password
 used to decrypt the private key.

cs_set_convert

Description Install or retrieve a user-defined conversion routine.

Syntax CS_RETCODE cs_set_convert(context, action, srctype, desttype, func)

```

CS_CONTEXT *context;
CS_INT      action;
CS_INT      srctype;
CS_INT      desttype;
CS_CONV_FUNC *func;

```

Parameters

context
 A pointer to a CS_CONTEXT structure. A CS_CONTEXT structure defines a Client-Library application context.

action
 One of the following symbolic values:

Value of <i>action</i>	<i>cs_set_convert</i>
CS_SET	Installs a conversion routine.
CS_GET	Retrieves the current conversion routine of this type.
CS_CLEAR	Clears the current conversion routine by replacing it with CS-Library's default conversion routine of this type.

srctype
 The datatype of the source data for the conversion.

desttype
 The datatype of the destination data.

func

A pointer to a CS_CONV_FUNC variable, which is a pointer to a custom conversion function. “Defining a custom conversion routine” on page 73 describes the prototype for a custom conversion function.

If a conversion routine is being installed, **func* points to the conversion routine that you wish to install.

If a conversion routine is being retrieved, cs_set_convert sets **func* to point to the currently installed conversion routine.

If a conversion routine is being cleared, pass **func* as NULL.

Note *func* represents a pointer to a pointer to a function. There are special requirements for passing this parameter. See the example code fragment under “Installing a custom conversion routine” on page 74.

Return value

cs_set_convert returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a cs_set_convert failure is an invalid parameter.

Usage

- An application can install custom conversion routines to convert data between:
 - Standard Open Client or Open Server datatypes
 - Standard and user-defined datatypes
 - User-defined datatypes
- Once a custom routine is installed for a particular conversion, the client/server libraries call the custom routine transparently whenever a conversion of the specified type is required.
- A Client-Library or Server-Library application creates a user-defined datatype by declaring it:

```
typedef CS_SMALLINT      EMPLOYEE_ID;
```

Because the Open Client routines ct_bind and cs_convert use integer symbolic constants to identify datatypes, it is often convenient for an application to declare a type constant for a user defined type. User-defined types must be defined as greater than or equal to CS_USERTYPE:

```
#define EMPLOYEE_ID_TYPE      CS_USERTYPE + 1;
```

To enable conversion between a user-defined type and standard CS-Library datatypes, an application can call `cs_set_convert` to install user-defined conversion routines for the new type.

- To clear a custom conversion routine, an application can call `cs_set_convert` with *action* as `CS_CLEAR` and *func* as `NULL`. `cs_set_convert` replaces the custom routine with CS-Library's default conversion routine of the appropriate type, if any.
- An application can call `cs_setnull` to define null substitution values for a user-defined type.

Defining a custom conversion routine

- A custom conversion routine is defined as follows:

```
CS_RETCODE CS_PUBLIC
convfunc(context, srcfmt, srcdata,
         destfmt, destdata, destlen)
CS_CONTEXT      *context;
CS_DATAFMT      *srcfmt;
CS_VOID         *srcdata;
CS_DATAFMT      *destfmt;
CS_VOID         *destdata;
CS_INT          *destlen;
```

where:

context is a pointer to a `CS_CONTEXT` structure.

srcfmt is a pointer to a `CS_DATAFMT` structure describing the source data. *srcfmt*→*maxlength* describes the actual length, in bytes, of the source data.

srcdata is a pointer to the source data.

destfmt is a pointer to a `CS_DATAFMT` structure describing the destination data. *destfmt*→*maxlength* describes the actual length, in bytes, of the destination data space.

destdata is a pointer to the destination data space.

destlen is a pointer to an integer. If the conversion is successful, the custom routine should set *destlen* to the number of bytes placed in *destdata*.

- `cs_config` is the only CS-Library, Client-Library, or Server-Library function that can be called from within a custom conversion routine.

- The following table lists the legal return values for a custom conversion routine. CS-Library will raise a CS-Library error if any value other than CS_SUCCEED is returned. Other values should be returned to indicate error conditions, as described in Table 2-14.
 - If the conversion routine returns a value listed in Table 2-14 other than CS_SUCCEED, then the application receives a Client-Library or CS-Library message that corresponds to the indicated error condition.
 - If the conversion routine returns a value that is not listed in Table 2-14, then the application receives an “Unknown return code” error message from Client-Library or CS-Library:

Table 2-14: Return values for a custom conversion routine

Return Value	To Indicate
CS_SUCCEED	Successful conversion.
CS_TRUNCATED	The conversion resulted in truncation.
CS_MEM_ERROR	A memory allocation failure has occurred.
CS_EBADXLT	Some characters could not be translated.
CS_ENOXLT	The requested translation is not supported.
CS_EDOMAIN	The source value is outside the domain of legal values for the datatype.
CS_EDIVZERO	Division by 0 is not allowed.
CS_EOVERFLOW	The conversion resulted in overflow.
CS_EUNDERFLOW	The conversion resulted in underflow.
CS_EPRECISION	The conversion resulted in loss of precision.
CS_ESCALE	An illegal scale value was encountered.
CS_ESYNTAX	The conversion resulted in a value which is not syntactically correct for the destination type.
CS_ESTYLE	The conversion operation was stopped due to a style error.

Installing a custom conversion routine

The following code demonstrates calling `cs_set_convert` to install a custom conversion routine, `MyConvert`, which converts from `CS_CHAR` to the user defined type indicated by `MY_USER_TYPE`. The code assumes that `MyConvert` is a custom conversion routine that has been defined correctly. (See “Defining a custom conversion routine” on page 73.) The program variable `myfunc` is used to pass the address of the conversion routine.

```
#define MY_USER_TYPE (CS_USER_TYPE + 2)
```

```

CS_CONV_FUNC p_conv_func;
p_conv_func = MyConvert;
if (cs_set_convert(context, CS_SET, CS_CHAR_TYPE, MY_USER_TYPE,
    &p_conv_func) != CS_SUCCEED)
{
    fprintf(stdout, "cs_set_convert(MY_USER_TYPE) failed!\n");
    (CS_VOID)ct_exit(context, CS_FORCE_EXIT);
    (CS_VOID)cs_ctx_drop(context);
    exit(1);
}

```

See also `cs_convert`, `cs_manage_convert`, `cs_setnull`, `ct_bind`

cs_setnull

Description Define a null substitution value to be used when binding or converting NULL data.

Syntax CS_RETURNCODE cs_setnull(context, datafmt, buffer, buflen)

```

CS_CONTEXT *context;
CS_DATAFMT *datafmt;
CS_VOID *buffer;
CS_INT buflen;

```

Parameters

context

A pointer to a CS_CONTEXT structure. `cs_setnull` defines a null substitution value for this context.

datafmt

A pointer to a CS_DATAFMT structure describing the datatype for which a null substitution value is being defined.

buffer

A pointer to the null substitution value. *buffer*'s datatype must match *datafmt*->type.

buflen

The length, in bytes, of *buffer*.

Return value

`cs_set_null` returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a cs_setnull failure include:

Usage

- A memory allocation error
- An invalid parameter
- If ANSI-style binds are in effect, CS-Library does not use null substitution values. To activate ANSI-style binds, an application sets the Client-Library property CS_ANSI_BINDS to CS_TRUE.
- When ANSI-style binds are not in effect and source data for a conversion is NULL, CS-Library sets the destination data to the predefined null substitution value for that destination type. For example, converting a NULL value of any type to a CS_CHAR destination results in an empty string.
- In a Client-Library application, null substitution values are defined at the context level. When a Client-Library connection is allocated, it picks up null substitution values from its parent context.
- When converting a NULL source value to a CS_CHAR or CS_BINARY destination variable, CS-Library first puts 0 bytes into the destination and then uses the *format* field of the CS_DATAFMT structure that describes the destination to determine whether to pad or null-terminate.
- To reinstate CS-Library’s original default null substitution value for a particular datatype, an application can call cs_setnull with *buffer* as NULL.
- CS-Library and Client-Library use the following default null substitution values:

Table 2-15: Default null substitution values

Destination Type	Null Substitution Value
CS_BINARY_TYPE	Empty array
CS_VARBINARY_TYPE	Empty array
CS_BIT_TYPE	0
CS_CHAR_TYPE	Empty string
CS_VARCHAR_TYPE	Empty string
CS_DATETIME_TYPE	8 bytes of zeros
CS_DATETIME4_TYPE	4 bytes of zeros
CS_TINYINT_TYPE	0
CS_SMALLINT_TYPE	0
CS_INT_TYPE	0
CS_DECIMAL_TYPE	0.0 (with default scale and precision)
CS_NUMERIC_TYPE	0.0 (with default scale and precision)

Destination Type	Null Substitution Value
CS_FLOAT_TYPE	0.0
CS_REAL_TYPE	0.0
CS_MONEY_TYPE	\$0.0
CS_MONEY4_TYPE	\$0.0
CS_BOUNDARY_TYPE	Empty string
CS_SENSITIVITY_TYPE	Empty string
CS_TEXT_TYPE	Empty string
CS_IMAGE_TYPE	Empty array

See also `cs_set_convert`, `cs_will_convert`

cs_strbuild

Description Construct native language message strings.

Syntax `CS_RETCODE cs_strbuild(context, buffer, buflen, resultlen, text, textlen [, formats, formatlen] [, arguments]);`

```
CS_CONTEXT *context;
CS_CHAR    *buffer;
CS_INT     buflen;
CS_INT     *resultlen;
CS_CHAR    *text;
CS_INT     textlen;
CS_CHAR    *formats; /* Optional */
CS_INT     formatlen; /* Optional */
<optional arguments>
```

Parameters *context* A pointer to a CS_CONTEXT structure.

buffer A pointer to the space in which `cs_strbuild` places the finished message. Note that the finished message is not null-terminated. An application must use *resultlen* to determine the length of the message placed in *buffer*.

buflen The length, in bytes, of the *buffer* data space.

resultlen

A pointer to an integer variable. `cs_strbuild` sets *resultlen* to the length, in bytes, of the string placed in *buffer*.

text

A pointer to the unfinished text of the message. The *text* string contains message text and placeholders for variables. A placeholder has the form *%integer!*, for example *%1!*, *%2!*, and so forth. The integer indicates which argument to substitute for a particular placeholder. Arguments are numbered from left to right.

textlen

The length, in bytes, of *text*. If *text* is null-terminated, pass *textlen* as `CS_NULLTERM`.

formats

A pointer to a string containing one `sprintf`-style format specifier for each place holder in the *text* string.

formatlen

The length, in bytes, of *formats*. If *formats* is null-terminated, pass *formatlen* as `CS_NULLTERM`.

arguments

The values which will be converted to character according to the *formats* string and substituted into the *text* string to produce the message that is placed in *buffer*.

There must be one argument for each place holder. The first value corresponds to the first format and the *%1!* placeholder, the second value corresponds to the second format and the *%2!* placeholder, and so forth.

If insufficient arguments are supplied, `cs_strbuild` generates unpredictable results.

If too many arguments are supplied, the excess arguments are ignored.

Return value

cs_str_build returns:

Returns	To Indicate
<code>CS_SUCCEEDED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.

Usage

- **cs_strbuild** builds a printable native language message string from a text containing place holders for values, a format string containing information on the types and appearances of the values, and a variable number of arguments that represent the values.

- Parameters in error messages can occur in different orders in different languages. `cs_strbuild` allows an application to construct error messages in a `sprintf`-like fashion to ensure easy translation of error messages from one language to another.

For example, consider an error message that informs the user of a misused keyword in a stored procedure. The message requires three arguments: the misused keyword, the line in which the keyword occurs, and the name of the stored procedure. In the U.S. English localization file, the message text appears as:

```
The keyword '%1!' is misused in line %2! of stored
procedure '%3!'.
```

In the Spanish localization file, the same message appears as:

```
En linea %2! de stored procedure '%3!', la palabra '%1!'
esta mal usado!
```

The `cs_strbuild` call for either of the above messages is:

```
cs_strbuild(context, &mybuffer, buflen,
            &resultlength, messagetext, CS_NULLTERM,
            "%s, %d, %s", CS_NULLTERM,
            keyword, linenum, sp_name);
```

The only difference is the content of *messagetext*.

- `cs_strbuild` format specifiers can be separated by other characters, or they can be adjacent to each other. This allows existing message strings in U.S. English to be used as format parameters. The first format specifier describes the `%1!` placeholder, the second describes the `%2!` placeholder, and so forth.

See also `cs_dt_crack`, `cs_dt_info`, `cs_locale`

cs_strcmp

Description Compare two strings using a specified sort order.

Syntax `CS_RETCODE cs_strcmp(context, locale, type, str1, len1, str2, len2, result)`

```
CS_CONTEXT *context;
CS_LOCALE *locale;
CS_INT type;
CS_CHAR *str1;
CS_INT len1;
```

```
CS_CHAR    *str2;  
CS_INTI    len2;  
CS_INT     *result;
```

Parameters*context*

A pointer to a CS_CONTEXT structure.

locale

A pointer to a CS_LOCALE structure. A CS_LOCALE structure contains locale information, including the collating sequence that cs_strcmp uses to define a sort order.

An application can call cs_locale with *type* as CS_LC_COLLATE or CS_SYB_SORTORDER to change the collating sequence in a CS_LOCALE structure.

locale can be NULL. If *locale* is NULL, cs_strcmp uses whatever localization information is defined in the *context* CS_CONTEXT structure. Localization information is always defined at the context level, because a CS_CONTEXT picks up default localization information when it is allocated.

type

The type of comparison to perform.

If *type* is CS_COMPARE, cs_strcmp performs a lexicographic comparison.

If *type* is CS_SORT, the values are compared as they would appear in a sorted list. It is possible for strings that are lexicographically equal to belong in different places in a sorted list.

str1

A pointer to the first string for the comparison.

len1

The length, in bytes, of **str1*. If **str1* is null-terminated, pass *len1* as CS_NULLTERM.

str2

A pointer to the second string for the comparison.

len2

The length, in bytes, of **str2*. If **str2* is null-terminated, pass *len2* as CS_NULLTERM.

result

A pointer to the result of the comparison. The following table lists the possible values for **result*:

Value of <i>*result</i>	To Indicate
<0	<i>str1</i> is lexicographically less than <i>str2</i> , or <i>str1</i> appears before <i>str2</i> in a sorted list.
0	<i>str1</i> is lexicographically equal to <i>str1</i> , or <i>str1</i> is identical to <i>str2</i> .
>0	<i>str1</i> is lexicographically greater than <i>str2</i> , or <i>str1</i> appears after <i>str2</i> in a sorted list.

Return value

cs_strcmp returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Usage

- `cs_strcmp` sets **result* to indicate the result of the comparison.
- Some languages contain strings that are lexicographically equal, according to a specific sort order, but contain different characters. Even though the strings are lexicographically equal, there is a standard order used when placing them into a sorted list.

An application can use `cs_strcmp` to compare strings either lexicographically or with respect to how they appear in a sorted list. For example, given a case-insensitive sort order that specifies that uppercase characters appear before lowercase characters in a sorted list:

- The strings “ABC” and “abc” are lexicographically equal
A call to `cs_strcmp` that compares “ABC” (as *str1*) and “abc” as (*str2*) with *type* as `CS_COMPARE` returns with *result* set to 0.
- “ABC” appears before “abc” in a sorted list
A call to `cs_strcmp` that compares “ABC” (as *str1*) and “abc” as (*str2*) with *type* as `CS_SORT` returns with *result* set to a value less than 0.
- `cs_strcmp` determines which sort order to use by examining **locale*, (or **context*, if *locale* is `NULL`).
- To change the sort order in a `CS_LOCALE` structure, an application calls `cs_locale` with *type* as `CS_LC_COLLATE` or `CS_SYB_SORTORDER`.

- To change the sort order in a CS_CONTEXT structure, an application must first set up a CS_LOCALE structure with the desired sort order and then call cs_config to set the CS_LOC_PROP property for the context.

See also `cs_cmp`, `cs_locale`, `cs_config`

cs_time

Description Retrieve the current date and time.

Syntax `CS_RETCODE cs_time(context, locale, buffer, buflen, outlen, daterec)`

```
CS_CONTEXT *context;  
CS_LOCALE *locale;  
CS_VOID *buffer;  
CS_INT buflen;  
CS_INT *outlen;  
CS_DATEREC *daterec;
```

Parameters

context

A pointer to a CS_CONTEXT structure.

locale

A pointer to a CS_LOCALE structure. A CS_LOCALE structure contains locale information, including formatting information that cs_time uses to create a current datetime string.

locale can be NULL. If *locale* is NULL, cs_time uses whatever localization information is defined in the CS_CONTEXT structure indicated by *context*. Localization information is always defined at the context level, because a CS_CONTEXT picks up default localization information when it is allocated.

buffer

A pointer to the space in which cs_time will place a character string representing the current date and time.

buffer is an optional parameter and can be passed as NULL. If *buffer* is NULL, *daterec* must be supplied.

buflen

The length, in bytes, of **buffer*.

If *buffer* is supplied and *buflen* indicates that **buffer* is not large enough to hold the current datetime string, *cs_time* sets **outlen* to the length of the datetime string and returns CS_FAIL.

If *buffer* is NULL, pass *buflen* as CS_UNUSED.

outlen

A pointer to an integer variable.

cs_time sets **outlen* to the length, in bytes, of the current datetime string.

If the string is larger than *buflen* bytes, an application can use the value of **outlen* to determine how many bytes are needed to hold the string.

If *buffer* is NULL, pass *outlen* as NULL.

If an application does not care about return length information, it can pass *outlen* as NULL.

daterec

A pointer to a CS_DATEREC structure in which *cs_time* will place the current date and time. Note that *cs_time* does not set the *datemsecond* and *datetzone* fields of the CS_DATEREC structure.

For more information on the CS_DATEREC structure, see *cs_dt_crack* in this chapter.

daterec is an optional parameter and can be passed as NULL. If *daterec* is NULL, *buffer* must be supplied.

Return value

cs_time returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Common reasons for a *cs_time* failure include:

- An invalid parameter.
- *buflen* indicates that the **buffer* data space is not large enough to hold the formatted datetime string.
- **cs_time** returns the current date and time either in character string format or in a CS_DATEREC structure, or both.

Usage

- cs_time formats the date and time according to locale information contained in **context*.

See also cs_config, cs_dt_crack, cs_dt_info, cs_locale

cs_validate_cb

Description A Client-Library callback routine, registered via ct_callback.

Syntax

```
typedef struct _cs_sslcertfield
{
    CS_VOID    *value;
    CS_INT     field_id;
    CS_INT     length;
} CS_SSLCERT_FIELD;

typedef struct _cs_sslcert
{
    CS_INT     field_count;
    CS_INT     extension_count;
    CS_UINT    start_date;
    CS_UINT    end_date;
    CS_SSLCERT_FIELD *fieldptr;
    CS_SSLCERT_FIELD *extensionptr;
} CS_SSLCERT;

typedef CS_INT (CS_PUBLIC * CS_CERT_CB) PROTOTYPE ((
    CS_VOID    *user_data,
    CS_SSLCERT *certptr,
    CS_INT     cert_count,
    CS_INT     valid
));
```

Parameters *certptr*
a pointer to an array of CS_SSLCERT which has cert_count elements. On return from the callback, all memory used is freed.

Note The array is not null terminated.

fieldptr
a pointer to field_count elements.

extensionptr
a pointer extension_count elements.

cs_will_convert

Description Indicate whether a specific datatype conversion is available in the Client/Server libraries.

Syntax CS_RETCODE cs_will_convert(context, srctype, desttype, result)

```
CS_CONTEXT *context;
CS_INT srctype;
CS_INT desttype;
CS_BOOL *result;
```

Parameters

context

A pointer to a CS_CONTEXT structure.

srctype

A symbolic constant representing the datatype of the source data (for example, CS_BYTE_TYPE, CS_CHAR_TYPE, and so forth).

desttype

A symbolic constant representing the datatype of the destination data.

result

A pointer to a boolean variable. cs_will_convert sets **result* to CS_TRUE if the datatype conversion is supported and CS_FALSE if the datatype conversion is not supported.

Return value

cs_willconvert returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Examples

```
/*
** ex_display_column()
*/
```

```
CS_RETCODE CS_PUBLIC
ex_display_column(context, colfmt, data, datalength,
indicator)
CS_CONTEXT *context;
CS_DATAFMT *colfmt;
CS_VOID *data;
CS_INT datalength;
CS_SMALLINT indicator;
```

```
{
    char          *null = "NULL";
    char          *nc   = "NO CONVERT";
    char          *cf   = "CONVERT FAILED";
    CS_DATAFMT   srcfmt;
    CS_DATAFMT   destfmt;
    CS_INT       olen;
    CS_CHAR      wbuf[MAX_CHAR_BUF];
    CS_BOOL      res;
    CS_INT       i;
    CS_INT       disp_len;

    if (indicator == CS_NULLDATA)
    {
        olen = strlen(null);
        strcpy(wbuf, null);
    }
    else
    {
        cs_will_convert(context, colfmt->datatype,
                       CS_CHAR_TYPE, &res);

        if (res != CS_TRUE)
        {
            olen = strlen(nc);
            strcpy(wbuf, nc);
        }
        else
        {
            srcfmt.datatype = colfmt->datatype;
            srcfmt.format   = colfmt->format;
            srcfmt.locale   = colfmt->locale;
            srcfmt.maxlength = datalength;

            destfmt.maxlength = MAX_CHAR_BUF;
            destfmt.datatype  = CS_CHAR_TYPE;
            destfmt.format    = CS_FMT_NULLTERM;
            destfmt.locale    = NULL;

            if (cs_convert(context, &srcfmt, data,
                          &destfmt, wbuf, &olen) != CS_SUCCEED)
            {
                olen = strlen(cf);
                strcpy(wbuf, cf);
            }
            else

```



```

    {
        /*
        ** output length include null
        ** termination
        */
        olen -= 1;
    }

}
}
fprintf(stdout, "%s", wbuf);

disp_len = ex_display_dlen(colfmt);
for (i = 0; i < (disp_len - olen); i++)
{
    fputc(' ', stdout);
}

return CS_SUCCEED;
}

```

Usage

- **cs_will_convert** allows an application to determine whether **cs_convert** or **ct_bind/ct_fetch** are capable of performing a specific conversion. When **cs_convert** is called to perform a conversion that it does not support, it returns **CS_FAIL** and generates a CS-Library error.
- **cs_convert** can convert between standard and user-defined datatypes. To enable these types of conversions, an application must install custom conversion routines via **cs_set_convert**. If a custom routine is supplied for a conversion, **cs_will_convert** indicates that the conversion is supported.

Datatype conversion chart

A chart listing the datatype conversions that **cs_convert** supports is included on the manual page for **cs_convert**. (See “Datatype Conversion Chart”.)

See also

cs_convert, **cs_set_convert**, **cs_setnull**

This chapter introduces Bulk-Library:

Topic	Page
Overview of Bulk-Library	89
Bulk-Library client programming	91
Bulk-Library gateway programming	98

Overview of Bulk-Library

Bulk-Library/C provides routines that allow Client-Library and Server-Library applications to use the Adaptive Server bulk-copy interface.

The Adaptive Server bulk-copy interface allows high-speed transfer of data between a client application's program variables and the server's database tables. It provides an alternative to the use of the SQL insert and select commands to transfer data.

Administrators can perform bulk copy using the bcp utility; programmers can use Bulk-Library to create customized bulk-copy tools. Bulk-Library also provides the necessary routines to enable bulk-copy support in an Open Server gateway application.

Note The Bulk-Library/C routines are for use with Open Client Client-Library and Open Server Server-Library applications. DB-Library provides its own bulk-copy interface, which is documented in the *Open Client DB-Library/C Reference Manual*.

Client-side and server-side routines

Bulk-Library contains client-side and server-side routines.

Client-side Bulk-Library routines

Client-side routines allow Client-Library programmers to execute bulk-copy commands from their programs. Client-side routines allow a program to:

- Transmit bulk-copy data to the remote server for database table population
- Extract the contents of a database table into program memory

Server-side Bulk-Library routines

Server-side routines are used with Open Server. Open Server programmers can use these routines together with the client-side routines to allow bulk-copy transfers through an Open Server gateway. A gateway server uses the client-side routines to obtain bulk-copy data from the remote server and server-side routines to forward the data to its own client. Any routine that requires a `SRV_PROC` (Open Server thread-control structure) pointer as an argument is a server-side routine.

The server-side Bulk-Library routines require the application to be linked with Server-Library and must be used together with the client-side routines.

Header files

The header file *bkpublic.h* contains Bulk-Library definitions and is required in all application source files that contain calls to Bulk-Library routines.

Client-Library applications that call Bulk-Library routines need to include only *bkpublic.h*, since *bkpublic.h* includes *ctpublic.h*. No harm is done if the application includes both files.

Gateway Open Server applications that call Bulk-Library routines need to include *bkpublic.h* in addition to the other include files required by Server-Library. *bkpublic.h* does not include any Open Server header files.

Linking with Bulk-Library

On most platforms, Bulk-Library is a separate library file and must be specified on the link line for the application. See the *Open Client/Server Programmer's Supplement* for compiling and linking instructions for your platform.

The CS_BLKDESC structure

All bulk-copy operations performed with Bulk-Library calls require a CS_BLKDESC structure. This structure is also called the *bulk-descriptor structure*. The bulk-descriptor structure is a hidden structure that controls a particular bulk-copy operation.

Applications allocate a bulk-descriptor structure with `blk_alloc` on page 104 and free the bulk descriptor's memory with `blk_drop` on page 126. The structure's internals are not documented, but the properties of the structure can be retrieved and modified with the `blk_props` on page 134 routine.

All Bulk-Library routines except for `blk_alloc` require a valid bulk-descriptor structure pointer as an input parameter.

The bulk-descriptor structure is considered a child structure of Client-Library's connection structure. Bulk-copy operations require the connection to interact with the remote server.

Bulk-Library client programming

Client-side Bulk-Library routines provide bulk-copy functionality to Client-Library programs. A Client-Library programmer may find bulk-copy useful if the application under development must exchange data with a non-database application, load data into a new database, or move data from one database to another.

A Client-Library application can call Bulk-Library routines to copy data either into a database table or out from a database table.

- Bulk-copy-in operations move data from the client machine into a database table and are typically used for database table population. For bulk copies into the database, Bulk-Library transmits tabular data over the network in its "raw" form. Bulk copies into the database can be considerably faster than embedding the data in equivalent SQL insert statements.

- Bulk-copy-out operations move data from a database table to the client program's memory space and are typically used for data extracts. For data extracts, bulk copy offers no performance advantage over the equivalent SQL select statements. However, the Bulk-Library interface may be more convenient for programmers.

Note Errors resulting from client-side Bulk-Library routines are reported as Client-Library errors. Applications should install a Client-Library message callback to handle these errors or handle them inline with `ct_diag`.

Bulk-Copy-In operations

An application can call Bulk-Library routines to copy data from program variables into a database table.

When copying into a database, the chief advantage of bulk copy over the SQL insert alternative is speed.

When copying data into a non-indexed table, the *high speed* version of bulk copy is used. Adaptive Server performs no data logging during high-speed transfers. If the system fails before the transfer is complete, no new data will remain in the database. Because high-speed transfer affects the recoverability of the database, it is enabled only when the Adaptive Server option `select into/bulkcopy` has been turned on. An application can call the Adaptive Server system procedure `sp_dboption` to turn this option on or use the Client Library connection property `CS_BULK_LOGIN`.

If the `select into/bulkcopy` option is not turned on and a user tries to copy data into a table that has no indexes, Adaptive Server generates an error message.

After a bulk-copy operation is complete, the System Administrator should dump the database to ensure its future recoverability.

When copying data into an indexed table, a slower version of bulk copy is automatically used, and row inserts are logged.

The Bulk-Copy-In process

A typical application follows these steps to perform a bulk-copy-in operation:

- 1 Initializes the application in the same way as for a Client-Library application and sets up Client-Library error handling. Bulk-Library reports errors generated by calls to client-side routines as Client-Library messages.
- 2 Allocates the connection structure to be used.
- 3 Calls `ct_con_props` to set the necessary properties to connect to the target server. In addition, the application must set the `CS_BULK_LOGIN` property to `CS_TRUE` to enable the connection to perform bulk copies.

Note Programmers can often tune the Tabular Data Stream™ (TDS) packet size to increase throughput. A packet size larger than the default usually increases performance. First, make sure that the Adaptive Server is configured to accept a larger TDS packet size, then set the `CS_PACKET_SIZE` connection property in your application. See the *Adaptive Server Enterprise System Administration Guide* for details on increasing the allowable network packet size and the *Open Client Client-Library/C Reference Manual* for details on connection properties.

- 4 Calls `ct_connect` to open the connection.
- 5 Calls `blk_alloc` to allocate a bulk-descriptor structure.
- 6 Calls `blk_init` to initialize the bulk-copy operation.
- 7 For each column in the target table, the application:
 - (Optional) Calls `blk_describe`. `blk_describe` returns a target column's description, allowing the application determine the column's datatype or size.
 - (Optional) Calls `blk_default`. `blk_default` returns a column's default value, if a default is defined by the table schema. An application can call `blk_bind` with **datalen* as 0 to indicate that the bulk-copy-in operation should use a column's default value.
 - Calls `blk_bind` to bind the variable to the target column. If data for the column will be transferred using `blk_textxfer`, the application must call `blk_bind` with *buffer* as NULL.

Columns can be bound either to scalar variables or to arrays. When columns are bound to scalar variables, each call to `blk_rowxfer_mult` transfers column values for a single row from the bound variables into the database. For array binding, an array is bound to each column, and multiple rows are transferred by each call to `blk_rowxfer_mult`. In either case, the application also binds *indicator* and *datalen* variables to the column as well. These are used to indicate the condition of the data to be transferred.

The discussion in this chapter assumes that array binding is not in effect. For more information about array binding, see `blk_bind` in Chapter 4, “Bulk-Library Routines”

8 Transfers the data.

While data remains to be transferred, the application places data into the program variables that are bound to the table columns, then calls `blk_rowxfer_mult` to transfer the row.

Before each call to `blk_rowxfer_mult`, for each bound column, the application sets *datalen* and *indicator* values to specify what value should be inserted:

<i>datalen</i> value	<i>indicator</i> value	Result
> 0	Any (is ignored).	<code>blk_rowxfer_mult</code> reads <i>datalen</i> bytes from <i>buffer</i> as the column value.
0	0	The column’s default value, if available, is inserted. If no default is available, NULL is inserted.
0	-1	NULL is inserted.

If the row contains columns whose data is being transferred in chunks, the application calls `blk_textxfer` in a loop for each column. Data being transferred via `blk_textxfer` must reside at the end of the row, following any bound columns.

The application can call `blk_done(CS_BLK_BATCH)`, if needed, to send a batch of rows. This call instructs the Adaptive Server to permanently save all rows transferred since the application’s last `blk_done` call.

9 Calls `blk_done(CS_BLK_ALL)` to send the last batch of rows and indicate that the bulk-copy operation is complete.

10 Calls `blk_drop` to deallocate the bulk-descriptor structure.

Note An application can call `blk_bind` between calls to `blk_rowxfer_mult` to specify a different program variable address or length.

Program structure for Bulk-Copy-In operations

Most applications use a program structure similar to the following pseudocode to perform a bulk-copy-in operation:

```

ct_con_props to set connection properties
ct_connect to open the connection
blk_alloc to allocate a CS_BLKDESC
blk_init to initiate the bulk copy

for each column
    (optional: blk_describe to get a description of
        the column)
    (optional: blk_default to get the column's default
        value)
    blk_bind to bind the column to a program
        variable, or to mark the column for transfer
        via blk_textxfer
endfor

while there's data to transfer
    if it's time to save a batch of rows
        blk_done(CS_BLK_BATCH)
    endif
    copy row values to program variables
    call blk_rowxfer_mult to transfer the row data

    if data is being transferred via blk_textxfer
        for each column to transfer
            while there's data for this column
                blk_textxfer to transfer a chunk of data
            endwhile
        endfor
    endif
endwhile
blk_done(CS_BLK_ALL)
blk_drop to deallocate the CS_BLKDESC

```

Bulk-Copy-Out operations

The bulk-copy-out process reads rows from the server and places the column values into program variables.

The Bulk-Copy-Out process

A typical application follows these steps to perform a bulk-copy-out operation:

- 1 Calls `ct_con_props` to set the required properties to open the connection.
- 2 Calls `ct_connect` to open the connection.
- 3 Calls `blk_alloc` to allocate a bulk-descriptor structure.
- 4 For each column of interest, the application:
 - (Optional) Calls `blk_describe` to retrieve a column's description. This step is necessary if an application lacks information about a column's datatype or size.
 - (Optional) Calls `blk_bind` to bind a program variable to the source column. If the data for a column will be transferred via `blk_textxfer`, call `blk_bind` with *buffer* as NULL.

Columns can be bound either to scalar variables or to arrays. When columns are bound to scalar variables, each call to `blk_rowxfer_mult` transfers column values for a single row into the bound variables into the database. For array binding, an array is bound to each column, and multiple column values are transferred into each array by each call to `blk_rowxfer_mult`.

The discussion in this chapter assumes that array binding is not used. For more information about array binding, see `blk_bind` in Chapter 4, "Bulk-Library Routines"

- 5 Transfers the data by calling `blk_rowxfer_mult` in a loop:

The application calls `blk_rowxfer_mult` repeatedly to transfer each row to program variables until `blk_rowxfer_mult` returns `CS_END_DATA`.

If the row contains columns whose data is transferred in chunks, the application calls `blk_textxfer` in a loop for each column. Data being transferred via `blk_textxfer` must reside at the end of the row, following any bound columns.

For example, suppose an application bulk-copies columns 1, 3, 5, 7, and 9 and must call `blk_textxfer` to copy columns 7 and 9. The application calls `blk_bind` once for each column, passing *buffer* as `NULL` for columns 7 and 9. After calling `blk_rowxfer_mult` to transfer a row from the table, the application must call `blk_textxfer` in a loop to copy the data for column 7 and then call `blk_textxfer` in another loop to copy the data for column 9.

- 6 Calls `blk_done(CS_BLK_ALL)` to indicate that the bulk-copy operation is complete.
- 7 Calls `blk_drop` to deallocate the bulk-descriptor structure.

Note An application can call `blk_bind` between calls to `blk_rowxfer_mult` to specify different program variable address or length.

Program structure for Bulk-Copy-Out operations

Most applications use a program structure similar to the following pseudocode to perform a bulk-copy-out operation:

```

ct_con_props to set connection properties
ct_connect to open the connection
blk_alloc to allocate a CS_BLKDESC
blk_init to initiate the bulk copy
for each column of interest
    (optional: blk_describe to get a description of
     the column)
    blk_bind to either bind the column to a program
    variable or to indicate that blk_textxfer will
    be used to transfer data for the column.
endfor
while there's data to transfer
    call blk_rowxfer_mult to transfer the row data
    pull data from program variables to a permanent
    location, if desired.
    if data is being transferred via blk_textxfer
        for each column to transfer
            while there's data for this column
                blk_textxfer to transfer a chunk of data
            endwhile
        endfor
    endif
endwhile
blk_done(CS_BLK_ALL)
blk_drop to deallocate the CS_BLKDESC

```

Copying to and from Secure SQL Server

Each row in a Secure SQL Server™ table has a sensitivity column, which contains the sensitivity label for the row. Secure SQL Server uses sensitivity labels to mediate access to data.

When bulk copying into or from a Secure SQL Server table, an application can choose whether or not to include the table's sensitivity column in the bulk-copy operation.

To include the sensitivity column, an application sets the `BLK_SENSITIVITY_LBL` property to `CS_TRUE`. `BLK_SENSITIVITY_LBL` has a default value of `CS_FALSE`, which means that by default the sensitivity column is not included.

Users copying into the sensitivity column must have the `bcpin_labels_role` activated on Secure SQL Server. If a user does not have this role, the bulk-copy operation will fail. See your Secure SQL Server documentation for more information on setting this role.

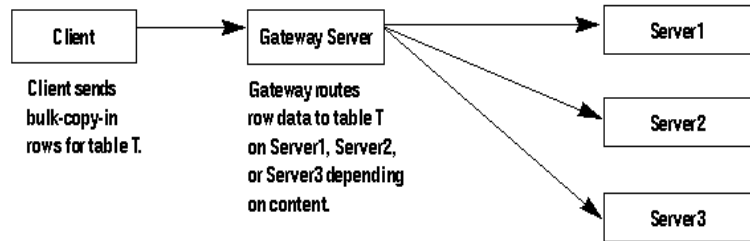
Bulk-Library gateway programming

The server-side Bulk-Library routines are designed to be used in gateways in conjunction with the client-side routines. Note that Open Server applications must have available a valid `CS_CONNECTION` structure (set up with Client-Library calls) to call Bulk-Library routines.

Open Server provides bulk-copy functionality that allows gateway Open Server applications to filter bulk-copy data. A gateway Open Server can examine each row of a bulk-copy operation and implement any of the following filters:

- Discard certain rows while keeping others,
- Send all rows to the remote server, or
- Route bulk-copy requests to multiple remote servers based on the row content, as shown in the diagram below.

Figure 3-1: Gateway routing bulk-copy requests



A gateway's client can issue two types of bulk requests, a *TDS text/image insert request* or a *TDS bulk-copy request*. In the case of a TDS text/image insert, the client simply wishes to send a text or image stream. In the case of a TDS bulk-copy request, the client is actually initiating a bulk-copy request. In both cases, the request handling involves processing both language (SRV_LANGUAGE) events and bulk (SRV_BULK) events.

An Open Server application processes both requests using two event handlers: SRV_LANGUAGE and SRV_BULK. Inside the SRV_LANGUAGE event handler, the application determines which kind of bulk request has been issued by the client and records this information internally. In addition, if the request is for bulk copy, the application allocates and initializes a bulk-descriptor structure. Inside the SRV_BULK handler, the application retrieves the request type and then processes the data accordingly.

The discussion in this section assumes that the gateway application is intended to accept both bulk-copy insert requests and text/image insert requests. For a description of how to handle text/image insert commands only, see the "Text and Image" topics page in the *Open Server Server-Library/C Reference Manual*.

Note Bulk-Library reports errors resulting from calls to server-side routines as Server-Library errors. Applications that call server-side Bulk-Library routines should install a Server-Library error handler to receive notification of these errors.

Inside the SRV_LANGUAGE event handler

If you intend for your gateway application to handle either type of bulk request, you must code the SRV_LANGUAGE event handler to parse for the phrase “insert bulk” or “writetext bulk.” These phrases indicate the following:

- The phrase “insert bulk” indicates the initiation of a bulk-copy request; the request handling will be started in the language handler and finished in the SRV_BULK handler.
- The phrase “writetext bulk” indicates that the client will issue a stream of text or image bytes to be handled in the SRV_BULK event handler.

“Insert Bulk” requests

The text of an “insert bulk” language request looks like this:

```
insert bulk tablename [with noddescribe]
```

where “with noddescribe” is optional.

In response, the SRV_LANGUAGE event handler should:

- 1 Record the bulk type internally by calling `srv_thread_props` with `cmd` set to `CS_SET`, `property` set to `SRV_T_BULKTYPE`, and `bufp` pointing to a value of `SRV_BULKLOAD`.
- 2 Continue parsing to extract the table name, which is an argument to the `blk_init` routine.
- 3 Allocate a bulk-descriptor structure, `CS_BLKDESC`, with a call to `blk_alloc`.
- 4 Initialize the client half of the exchange with a call to `blk_init`.
- 5 If “with noddescribe” is specified, it means that this data is part of a batch, and the table into which the bulk data will be loaded has already been described. The application need not call `blk_srvinit` a second time.

If “with noddescribe” is not specified, initialize the server half of the exchange with a call to `blk_srvinit`.

“Writetext Bulk” requests

The text of a “writetext bulk” language request looks like this:

```
writetext bulk dbname.tblname.colname textptr  
[timestamp=timestamp] [with log]
```

where the timestamp and logging indicator are optional.

In response, the SRV_LANGUAGE event handler should:

- 1 Record the bulk type internally by calling `srv_thread_props` with `cmd` set to `CS_SET`, `property` set to `SRV_T_BULKTYPE`, and `bufp` pointing to a value of `SRV_TEXTLOAD` or `SRV_IMAGELOAD`.
- 2 Continue parsing to extract the object name, which is generally of the form “`dbname.tblname.colname`”. This name can then be stored in the `name` and `namelen` fields of a `CS_IODESC` structure, which can later be used in the `SRV_BULK` event handler as an argument to `ct_data_info`, if the data stream is being passed on to a server in a gateway application.
- 3 Continue parsing to extract the text pointer, which will appear as a large hexadecimal number. Once converted from a character string to an actual `CS_BINARY` value, the text pointer and its length are stored in the `textptr` and `textptrlen` fields of the `CS_IODESC` structure.
- 4 Continue parsing to extract the timestamp, which, if present, will appear as “`timestamp = large_hexadecimal_number`”. Once converted from a character string to an actual `CS_BINARY` value, the timestamp and its length can be stored in the `timestamp` and `timestamplen` fields of the `CS_IODESC` structure.
- 5 Finally, parse to extract the logging indicator, which, if present, will appear as “with log”. If this indicator is present, the `log_on_update` field of the `CS_IODESC` structure should be set to `CS_TRUE`.

Inside the SRV_BULK event handler

Inside the `SRV_BULK` event handler, the application must respond to the bulk request that triggered the handler. However, its response depends on which type of bulk request the client issued. The application retrieves the request type by calling `srv_thread_props` with `cmd` set to `CS_GET` and `property` set to `SRV_T_BULKTYPE`.

If the request type is `SRV_TEXTLOAD` or `SRV_IMAGELOAD`, the application reads the text or image data from the client in chunks, using the `srv_text_info` and `srv_get_text` routines. For details, see the “Text and Image” topics page in the *Open Server Server-Library/C Reference Manual*.

If the request type is `SRV_BULKLOAD`, the application processes the bulk-copy rows using a combination of client-side and server-side routines. To process the bulk-copy rows, the `SRV_BULK` event handler should:

- 1 Call `blk_rowalloc` to allocate a `CS_BLK_ROW` structure.
The `CS_BLK_ROW` structure is a hidden structure that holds formatted bulk-copy rows sent from the client.
- 2 Call `blk_getrow` to retrieve the formatted row from the client. This call retrieves all column data except columns of type text, image, sensitivity, or boundary. The gateway can process these later. If the row contains text, image, sensitivity, or boundary data, `blk_getrow` returns `CS_BLK_HASTEXT`. Otherwise, it returns `CS_SUCCEED`. If there are no more rows, the bulk-copy operation is complete and `blk_getrow` returns `CS_END_DATA`.
- 3 If the gateway must examine the row content (for example, to route rows to particular remote servers or reject data), it calls `blk_colval` to examine the value of each column in the bulk row.
- 4 Call the client-side routine `blk_sendrow` to send the formatted rows to the remote server.
- 5 If an incoming bulk row contains text, image, sensitivity, or boundary data, the server portion of the gateway calls `blk_gettext` to retrieve the row's text, image, sensitivity, or boundary portion. The handler calls the client-side routine `blk_sendtext` to send it on to the remote server.
- 6 Call `blk_rowdrop` to deallocate the `CS_BLK_ROW` structure allocated by `blk_rowalloc`.
- 7 Call the client-side routine `blk_done` to indicate that the batch or bulk-copy operation is complete.
- 8 Call `blk_drop` to deallocate the bulk-descriptor structure.

Example

The online Open Server sample `ctosdemo.c` includes code to process bulk-copy requests.

Bulk-Library Routines

This chapter contains a reference page for each Bulk-Library routine.

List of Bulk-Library routines

Routine	Description
blk_alloc	Allocate a CS_BLKDESC structure.
blk_bind	Bind a program variable and a database column.
blk_colval	Server-side routine to obtain the column value from a formatted bulk copy row.
blk_default	Retrieve a column's default value.
blk_describe	Retrieve a description of a database column.
blk_done	Mark a complete bulk copy operation or a complete bulk copy batch.
blk_drop	Deallocate a CS_BLKDESC structure.
blk_getrow	Server-side routine to retrieve and store a formatted bulk copy row.
blk_gettext	Server-side routine to retrieve the text, image, sensitivity, or boundary portion of an incoming bulk copy formatted row.
blk_init	Initiate a bulk copy operation.
blk_props	Set or retrieve bulk descriptor structure properties.
blk_rowalloc	Server-side routine to allocate space for a formatted bulk copy row.
blk_rowdrop	Server-side routine to free space previously allocated for a formatted bulk copy row.
blk_rowxfer	Transfer one or more rows during a bulk copy operation without specifying or receiving a row count.
blk_rowxfer_mult	Transfer one or more rows during a bulk copy operation.

Routine	Description
blk_sendrow	Server-side routine to send a formatted bulk copy row obtained from blk_getrow.
blk_sendtext	Server-side routine to send text, image, sensitivity, or boundary data in a formatted bulk copy row obtained from blk_sendtext.
blk_srvinit	Server-side routine to copy descriptions of server table columns to the client, if required.
blk_textxfer	Transfer a column's data in chunks during a bulk copy operation.

blk_alloc

Description

Allocate a CS_BLKDESC structure.

Syntax

CS_RETCODE blk_alloc(connection, version, blk_pointer)

```
CS_CONNECTION  *connection;
CS_INT         version;
CS_BLKDESC     **blk_pointer;
```

Parameters

connection

A pointer to a CS_CONNECTION structure that has been allocated with ct_con_alloc and opened with ct_connect. A CS_CONNECTION structure contains information about a particular client/server connection.

The connection must not have any pending results.

version

The intended version of Bulk-Library behavior. During initialization, *version*'s value is checked for compatibility with Client-Library's version level. *version* can take the following values:

Value	Meaning	Compatible Client-Library Version Level(s)
BLK_VERSION_100	Version 10.0 behavior	CS_VERSION_110, CS_VERSION_100
BLK_VERSION_110	Version 11.0 behavior	Same as BLK_VERSION_100
BLK_VERSION_120	Version 12.0 behavior.	Same as BLK_VERSION_100, 110
BLK_VERSION_125	Version 12.5 behavior.	Same as BLK_VERSION_100, 110, 120

Note BLK_VERSION_100 can only be used with Open Client/Server versions 11.x and higher, regardless of whether the context/ctlib is initialized to CS_VERSION_100 or CS_VERSION_110.

The application's Client-Library version level is determined by the call to `ct_init` that initializes the connection's parent context structure.

blk_pointer

The address of a pointer variable. `blk_alloc` sets *blk_pointer* to the address of a newly allocated CS_BLKDESC structure.

In case of error, `blk_alloc` sets *blk_pointer* to NULL.

Return value

`blk_alloc` returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

The most common reason for a `blk_alloc` failure is a lack of adequate memory.

Examples

```
/*
** BulkCopyIn()
** Ex_tabname is globally defined.
*/
CS_STATIC CS_RETCODE
BulkCopyIn(connection)
```

```
CS_CONNECTION    *connection;
{
    CS_BLKDESC     *blkdesc;
    CS_DATAFMT     datafmt;      /* variable descriptions */
    Blk_Data       *dptr;        /* data for transfer */
    CS_INT         datalen[5];   /* variable data length */
    CS_INT         len;
    CS_INT         numRows;

    /*
    ** Ready to start the bulk copy in now that all the
    ** connections have been made and have a table name.
    ** Start by getting the bulk descriptor and
    ** initializing.
    */
    if (blk_alloc(connection, BLK_VERSION_100, &blkdesc)
        != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_alloc() failed");
        return CS_FAIL;
    }
    if (blk_init(blkdesc, CS_BLK_IN,
                Ex_tabname, strlen(Ex_tabname)) == CS_FAIL)
    {
        ex_error("BulkCopyIn: blk_init() failed");
        return CS_FAIL;
    }
    /*
    ** Bind the variables to the columns and send the rows,
    ** and then clean up.
    */
    ...CODE DELETED.....

    return CS_SUCCEED;
}
```

Usage

- A CS_BLKDESC structure, also called a *bulk-descriptor structure*, is the control structure for sending and receiving bulk-copy data. It is a hidden structure that contains information about a particular bulk-copy operation.
- Before calling blk_alloc, an application must call the Client-Library routines ct_con_alloc and ct_connect to allocate a CS_CONNECTION structure and open the connection.
- blk_alloc must be the first routine called in a bulk-copy operation.

- Multiple CS_BLKDESC and CS_COMMAND structures can be allocated on a connection, but only one CS_BLKDESC or CS_COMMAND structure can be active at a time. For more information, see blk_init on page 131 in this chapter.
- To deallocate a CS_BLKDESC structure, an application can call blk_drop.

See also

blk_drop, blk_init, ct_con_alloc, ct_connect

blk_bind

Description

Bind a program variable to a database column.

Syntax

```
CS_RETCODE blk_bind(blkdesc, colnum, datafmt, buffer,
                    datalen, indicator)
```

```
CS_BLKDESC  *blkdesc;
CS_INT      colnum;
CS_DATAFMT  *datafmt;
CS_VOID     *buffer;
CS_INT      *datalen;
CS_SMALLINT *indicator;
```

Parameters

blkdesc

A pointer to the CS_BLKDESC that is serving as a control block for the bulk-copy operation. blk_alloc allocates a CS_BLKDESC structure.

colnum

The number of the column to bind to the program variable.

The first column in a table is column number 1, the second is number 2, and so forth.

datafmt

A pointer to the CS_DATAFMT structure that describes the program variable to bind to the column.

Table 4-1 lists the fields in **datafmt* that are used by blk_bind and contains general information about the fields. blk_bind ignores fields that it does not use:

Table 4-1: Fields in the CS_DATAFMT structure for blk_bind

Field Name	When Used	Set the Field To
<i>name</i>	Not used.	Not applicable.
<i>namelen</i>	Not used.	Not applicable.

Field Name	When Used	Set the Field To
<i>datatype</i>	Always.	<p>A type constant (CS_XXX_TYPE) representing the datatype of the program variable.</p> <p>All type constants listed on the “Types” topics page in the <i>Open Client Client-Library/C Reference Manual</i> are valid.</p> <p>Open Client user-defined types are not valid.</p> <p>blk_bind supports a wide range of type conversions, so <i>datatype</i> can be different from the column’s type. For instance, by specifying a variable type of CS_FLOAT_TYPE, a <i>money</i> column can be bound to a CS_FLOAT program variable. blk_rowxfer_mult on page 143 or blk_rowxfer on page 140 perform appropriate conversions when transferring data. For a list of the data conversions provided by Client-Library, see cs_convert on page 25 in Chapter 2, “CS-Library Routines”</p> <p>If <i>datatype</i> is CS_BOUNDARY_TYPE or CS_SENSITIVITY_TYPE, the <i>*buffer</i> program variable must be of type CS_CHAR.</p>
<i>format</i>	When binding to character- or binary-type destination variables during copy-out operations; otherwise, CS_FMT_UNUSED.	<p>For variable-length datatypes, the setting is a bit mask that indicates the format of data to be read or the format to write data in.</p> <p>For bulk copy out, the format flags are the same as for ct_bind.</p> <p>For bulk copy in, the only format flag is CS_BLK_ARRAY_MAXLEN. For more information on the use of this flag, see “Array binding” on page 116.</p>

Field Name	When Used	Set the Field To
<i>maxlength</i>	When binding to a variable length datatype. When binding to a fixed-length datatype, <i>maxlength</i> is ignored.	The maximum length of the <i>*buffer</i> program variable. When binding character or binary variables, <i>maxlength</i> must describe the total maximum length of the program variable, including any space required for special terminating bytes, such as a null terminator. During a bulk-copy-in operation, <i>maxlength</i> specifies the maximum length of the data that will be copied from the <i>*buffer</i> program variable. During a bulk-copy-out operation, <i>maxlength</i> is the length of the <i>*buffer</i> program variable.
<i>scale</i>	Only when binding to numeric or decimal variables.	The scale of the program variable. If the source data is the same type as the destination, then <i>scale</i> can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for <i>scale</i> from the source data. <i>scale</i> must be less than or equal to <i>precision</i> .
<i>precision</i>	Only when binding numeric or decimal destinations.	The precision of the program variable. If the source data is the same type as the destination, then <i>precision</i> can be set to CS_SRC_VALUE to indicate that the destination should pick up its value for <i>precision</i> from the source data. <i>precision</i> must be greater than or equal to <i>scale</i> .
<i>status</i>	Not used.	Not applicable.

Field Name	When Used	Set the Field To
<i>count</i>	Always.	<p><i>count</i> is the number of rows to transfer per <code>blk_rowxfer_mult</code> on page 143 or <code>blk_rowxfer</code> on page 140 call. If <i>count</i> is greater than 1, array binding is considered to be in effect.</p> <p>During a bulk-copy-out operation, if <i>count</i> is larger than the number of available rows, only the available rows are copied.</p> <p><i>count</i> must have the same value for all columns being transferred, with one exception: an application can intermix <i>counts</i> of 0 and 1. This is because when <i>count</i> is 0, 1 row is transferred.</p>
<i>usertype</i>	Not used.	Not applicable.
<i>locale</i>	If supplied, <i>locale</i> is used. Otherwise, default localization applies.	A pointer to a <code>CS_LOCALE</code> structure containing locale information for the <i>*buffer</i> program variable.

buffer

The address of the program variable to be bound to the column specified by *colnum*.

For a bulk copying in, **buffer* is the program variable from which `blk_rowxfer_mult` copies the data.

For bulk copying out, **buffer* is the program variable in which `blk_rowxfer_mult` places the copied data. If *datafmt*→*maxlength* indicates that **buffer* is not large enough to hold the copied data, `blk_rowxfer_mult` truncates the data at row transfer time. If this occurs, Bulk-Library sets **indicator* to the actual length of the available data.

A NULL *buffer* indicates that data for the column will be transferred using the `blk_textxfer` routine.

datalen

A pointer to the length, in bytes, of the **buffer* data.

For bulk-copy-in operations:

- If **buffer* is not NULL, **datalen* represents the actual length of the data contained in the **buffer* program variable. An application must set this length before calling `blk_rowxfer_mult` or `blk_rowxfer` to transfer rows. In case of variable-length data, the length may be different for each row. If the data is fixed-length, **datalen* can be `CS_UNUSED`. If **datalen* is 0, the value of **indicator* is used to determine whether the column's default value or a NULL should be inserted—see Table 4-2 on page 114 for details.
- If **buffer* is NULL (indicating that the data will be transferred with `blk_textxfer`), **datalen* indicates the total length of the value to be transferred.

For bulk-copy-out operations:

- **datalen* represents the actual length of the data copied to **buffer*. `blk_rowxfer_mult` or `blk_rowxfer` sets **datalen* each time it is called to transfer a row.

indicator

A pointer to a `CS_INT` variable, or for array binding, an array of `CS_INT`. At row-transfer time, `blk_rowxfer_mult` or `blk_rowxfer` read the indicator's contents to determine certain conditions about the bulk-copy data. See the “List of Bulk-Library routines” on page 103 section for details.

Return value

`blk_bind` returns:

Returns	To Indicate
<code>CS_SUCCEED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.

`blk_bind` returns `CS_FAIL` if the application has not called `blk_init` to initialize the bulk-copy operation.

Examples

```
/*
** BulkCopyIn()
** BLKDATA and DATA_END are defined in the bulk copy
** example program.
*/

CS_STATIC CS_RETCODE
```

```
BulkCopyIn(connection)
CS_CONNECTION *connection;
{
    CS_BLKDESC *blkdesc;
    CS_DATAFMT datafmt; /* variable descriptions */
    Blk_Data *dptr; /* data for transfer */
    CS_INT datalen[5]; /* variable data length */
    CS_INT len;
    CS_INT numRows;

    /*
    ** Ready to start the bulk copy in now that all the
    ** connections have been made and have a table name.
    ** Start by getting the bulk descriptor initializing.
    */
    ...CODE DELETED.....

    /*
    ** Bind the variables to the columns and
    ** transfer the data.
    */
    datafmt.locale = 0;
    datafmt.count = 1;
    dptr = BLKDATA;
    while (dptr->pub_id != DATA_END)
    {
        datafmt.datatype = CS_INT_TYPE;
        datafmt.maxlength = sizeof(CS_INT);
        datalen[0] = CS_UNUSED;

        if (blk_bind(blkdesc, 1, &datafmt, &dptr->pub_id,
                    &datalen[0], NULL) != CS_SUCCEED)
        {
            ex_error("BulkCopyIn: blk_bind(1) failed");
            return CS_FAIL;
        }
        datafmt.datatype = CS_CHAR_TYPE;
        datafmt.maxlength = MAX_PUBNAME - 1;
        datalen[1] = strlen(dptr->pub_name);
        if (blk_bind(blkdesc, 2, &datafmt, dptr->pub_name,
                    &datalen[1], NULL) != CS_SUCCEED)
        {
            ex_error("BulkCopyIn: blk_bind(2) failed");
            return CS_FAIL;
        }
    }
}
```

```

    datafmt.maxlength = MAX_PUBCITY - 1;
    datalen[2] = strlen(dptra->pub_city);
    if (blk_bind(blkdesc, 3, &datafmt, dptra->pub_city,
        &datalen[2], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_bind(3) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_PUBST - 1;
    datalen[3] = strlen(dptra->pub_st);
    if (blk_bind(blkdesc, 4, &datafmt, dptra->pub_st,
        &datalen[3], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_bind(4) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_BIO - 1;
    datalen[4] = strlen((char *)dptra->pub_bio);
    if (blk_bind(blkdesc, 5, &datafmt, dptra->pub_bio,
        &datalen[4], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_bind(5) failed");
        return CS_FAIL;
    }
    if (blk_rowxfer (blkdesc) == CS_FAIL)
    {
        ex_error("BulkCopyIn: blk_rowxfer() failed");
        return CS_FAIL;
    }
    dptra++;
}

/* Mark the operation complete and then clean up */
...CODE DELETED.....

return CS_SUCCEED;
}

```

Usage

- `blk_bind` is a client-side routine.
- `blk_bind` binds program variables to table columns in the database. Once variables are bound, subsequent calls to `blk_rowxfer_mult` copy row data between the database and the bound variables. The copy direction is determined by the application's earlier call to `blk_init`.

- When copying into a database, an application must call `blk_bind` once for each column in the database table. When copying out, an application need not call `blk_bind` for columns in which it has no interest.
- To indicate that a column value will be transferred via `blk_textxfer`, an application calls `blk_bind` with *buffer* as NULL. A typical application will use `blk_textxfer` to transfer large text or image values.

If a text, image, boundary, or sensitivity datatype column is marked for transfer via `blk_textxfer`, all subsequent columns of these types must also be marked for transfer via `blk_textxfer`. For example, an application cannot mark the first text column in a row for transfer via `blk_textxfer` and then bind a subsequent text column to a program variable.

- An application can call `blk_bind` in between calls to `blk_rowxfer_mult` to reflect changes in a variable's address or length. If an application calls `blk_bind` multiple times for a single column or variable, only the last binding takes effect.
- An application can call `blk_describe` to initialize a `CS_DATAFMT` structure that describes the format of a particular column.

blk_bind for Bulk-Copy-In operations

The following table summarizes `blk_bind` usage when used for bulk-copy-in operations. For information on *datafmt* fields, see Table 4-1 on page 107.

Table 4-2: *blk_bind* parameter values for bulk copy in

When calling <code>blk_bind</code> To	<i>buffer</i> is	<i>datalen</i> is	<i>*indicator</i> is
Bind to a scalar or array variable from which <code>blk_rowxfer_mult</code> will read column values.	The address of a program variable or array.	A pointer to a variable or array that indicates the length of the values to be read from <i>*buffer</i> . <ul style="list-style-type: none"> • If <i>*datalen</i> is greater than 0, <i>*datalen</i> values are read from <i>*buffer</i> and sent as the column value. • When <i>*datalen</i> is 0, the value of <i>*indicator</i> is used to determine whether the column's default value (if any) or NULL should be inserted. 	The address of a variable or array that supplies indicator values for the column. <p><i>*indicator</i> is only considered when <i>*datalen</i> is 0:</p> <ul style="list-style-type: none"> • If <i>*indicator</i> is 0, the column's default value (if available) is inserted. If no default value is available, a NULL is inserted. • If <i>*indicator</i> is -1, NULL is always inserted.

When calling <code>blk_bind</code> To	buffer is	datalen is	*indicator is
Indicate that a column value will be transferred using <code>blk_textxfer</code> .	NULL	The total length of the data that will be sent using <code>blk_textxfer</code> . In this case, <code>datafmt->maxlength</code> is ignored.	Ignored.

`blk_bind` for Bulk-Copy-Out operations

The following table summarizes `blk_bind` usage when used for bulk-copy-out operations. For information on `datafmt` fields, see Table 4-1 on page 107:

Table 4-3: `blk_bind` parameter values for bulk copy out

When calling <code>blk_bind</code> To	buffer is	*datalen is	*indicator is
Bind to a scalar or array variable into which <code>blk_rowxfer_mult</code> will write column values.	The address of a program variable or array.	A pointer to a variable or array where <code>blk_rowxfer_mult</code> on page 143 places the length of the values written to <code>*buffer</code> .	The address of a variable or array that supplies indicator values for the column. <code>blk_rowxfer_mult</code> sets <code>*indicator</code> as follows: <ul style="list-style-type: none"> • -1 indicates the data is null. • 0 indicates good data. • A value greater than 0 indicates truncation occurred. The value is the actual length of the available data.
Indicate that a column value will be transferred using <code>blk_textxfer</code> .	NULL	Ignored. In this case, <code>datafmt->maxlength</code> represents the length of the <code>*buffer</code> data space.	Ignored.

Specifying Null values for Bulk Copy into the database

- When copying in, an application can instruct `blk_rowxfer_mult` to use a column's default value by setting `*datalen` to 0 and `*indicator` to 0 before calling `blk_rowxfer_mult`. If no default value is defined for the column, `blk_rowxfer_mult` inserts a NULL value.
- To instruct `blk_rowxfer_mult` to insert a NULL regardless of a column's default value, set `*datalen` to 0 and `*indicator` to -1 before calling `blk_rowxfer_mult`.

Clearing bindings

- To clear a binding, call `blk_bind` with `buffer`, `datafmt`, `datalen`, and `indicator` as `NULL`. Otherwise, bindings remain in effect until an application calls `blk_done` with `type` as `CS_BLK_ALL` to indicate that the bulk-copy operation is complete.
- To clear all bindings, pass `colnum` as `CS_UNUSED`, with `buffer`, `datafmt`, `datalen`, and `indicator` as `NULL`. An application typically clears all bindings when it needs to change the count that is being used for array binding.

Array binding

- Array binding is the process of binding a column to an array of program variables. At row-transfer time, multiple rows' worth of the column are transferred either to or from the array of variables with a single `blk_rowxfer_mult` call. An application indicates array binding by setting `datafmt->count` to a value greater than 1.
- Array binding works differently for bulk-copy-in and bulk-copy-out operations.
- For bulk-copy-in operations that use array binding, you must call `blk_bind` with `buffer`, `datalen`, and `indicator` pointing to arrays. Each length and indicator variable describes the corresponding data in the buffer array. For fixed-length data, `buffer` is always a pointer to an array of fixed-length values. For variable-length data (specifically character or binary data), `buffer` is a pointer to an array of bytes. In the latter case, the packing of values can be *loose* or *dense*. The application specifies the packing method for each column by setting flags in the `datafmt->format` field:
 - Setting the `BLK_ARRAY_MAXLEN` bit in `datafmt->format` specifies *loose* packing of values in the array. `blk_rowxfer_mult` retrieves the value *i* by reading `datalen[i-1]` bytes starting at the byte position computed as:

$$(i - 1) * datafmt->maxlength$$

- If the `BLK_ARRAY_MAXLEN` bit is not set in `datafmt->format`, column values must be densely packed for `blk_rowxfer_mult`. Each value must be placed in the column array immediately after the previous value, without padding. `blk_rowxfer_mult` gets value *i* by reading `datalen[i-1]` bytes starting at the byte position computed as:

$$datalen[i-2] + datalen[i-3] + \dots + datalen[0]$$

In other words, the first value starts at 0, the second at `datalen[0]`, the third at `datalen[1] + datalen[0]`, and so forth.

For example, consider a character column that will receive the values “girl”, “boy”, “man”, and “woman”, and assume that this column is bound with `datafmt->maxlength` passed as 7. With loose array binding, the `buffer` and `datalen` contents would be:

```
buffer: girl   boy   man   woman
        0     7   14   21
datalen: 4, 3, 3, 5
```

With densely packed array binding, the `buffer` and `datalen` contents would be:

```
buffer: girlboymanwoman
        0  4  7 10
datalen: 4, 3, 3, 5
```

- For bulk-copy-out operations, array binding performed with `blk_bind` works the same as array binding performed with `ct_bind`. Column arrays for bulk-copy-out are always loosely packed.
- While using array binding during a bulk-copy-out operation, it is possible for conversion, memory, or truncation errors to occur while `blk_rowxfer_mult` is writing to the destination arrays. In this case, `blk_rowxfer_mult` writes a partial result to the destination arrays and returns `CS_ROW_FAIL`.
- If array binding is in effect (for either direction), an application cannot use `blk_textxfer` to transfer data.

See also

`blk_describe`, `blk_default`, `blk_init`

blk_colval

Description

Server-side routine to obtain the column value from a formatted bulk-copy row.

Syntax

```
CS_RETCODE blk_colval(srvproc, blkdesc, rowp, colnum,
                      valuep, valuelen, outlenp)
```

```
SRV_PROC      *srvproc;
CS_BLKDESC    *blkdesc;
CS_BLK_ROW    *rowp;
CS_INT        colnum;
CS_VOID       *valuep;
```

CS_INT valuelen;
CS_INT *outlen;

Parameters

srvproc

A pointer to the SRV_PROC structure associated with the client sending the bulk-copy row. It contains all the information that Server-Library uses to manage communications and data between the Open Server application and the client.

blkdescp

A pointer to a CS_BLKDESC structure containing information about bulk-copy data. This structure must have been previously allocated with a call to blk_alloc and initialized with a call to blk_init. This structure is used to interpret incoming formatted bulk-copy rows.

rowp

A pointer to the CS_BLK_ROW structure filled in by a prior call to blk_getrow.

The CS_BLK_ROW structure is a hidden structure that holds formatted bulk-copy rows sent from the client.

colnum

The column number of the column of interest. Column numbers start at 1.

valuep

A pointer to the application buffer in which the column value from the bulk-copy row is placed.

valuelen

The size, in bytes, of the buffer to which *valuep* points.

outlen

A pointer to a CS_INT variable. blk_colval sets *outlen to the size, in bytes, of the column data.

Return value

blk_colval returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Usage

- blk_colval is a server-side routine. After getting the value of a specified column from a formatted bulk-copy row, it stores the value in an application buffer.
- This routine performs no implicit data conversion. Use cs_convert to convert the data.

- To examine the column value after a call to `blk_colval`, the application must know the column's datatype before making the call.
- An Open Server application cannot use this routine to retrieve text, image, sensitivity, or boundary columns. Use `blk_gettext` to retrieve such columns.

See also `blk_getrow`, `blk_gettext`

blk_default

Description	Retrieve a column's default value.
Syntax	<pre>CS_RETCODE blk_default(blkdesc, colnum, buffer, buflen, outlen) CS_BLKDESC *blkdesc; CS_INT colnum; CS_VOID *buffer; CS_INT buflen; CS_INT *outlen;</pre>
Parameters	<p><i>blkdesc</i> A pointer to the <code>CS_BLKDESC</code> that serves as a control block for the bulk-copy operation. <code>blk_alloc</code> allocates a <code>CS_BLKDESC</code> structure.</p> <p><i>colnum</i> The number of the column of interest. The first column in a table is column number 1, the second is number 2, and so forth.</p> <p><i>buffer</i> A pointer to the space in which <code>blk_default</code> will place the default value.</p> <p><i>buflen</i> The length, in bytes, of the <i>buffer</i> data space.</p> <p><i>outlen</i> A pointer to an integer variable. If supplied, <code>blk_default</code> sets <i>outlen</i> to the length, in bytes, of the default value. If the default value is larger than <i>buflen</i> bytes, an application can use the value of <i>outlen</i> to determine how many bytes are needed to hold the value.</p>
Return value	<code>blk_default</code> returns:

Returns	To Indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

blk_default returns CS_FAIL if the application has not called blk_init to initialize the bulk-copy operation.

Usage

- blk_default is a client-side routine.
- An application can call blk_default to find out whether a default value is defined for a particular target column, and, if so, what the default value is. This information can be useful while preparing to bulk copy rows into a database. The application can set **datalen* and **indicator* values to specify whether a column's default value should be used. (*datalen* and *indicator* are the addresses of program variables that were bound to the column with blk_bind). See "Specifying Null values for Bulk Copy into the database" on page 115 for more information.
- If the column of interest does not have a default value, blk_default sets **outlen* to CS_NO_DEFAULT and returns CS_SUCCEED.
- An application can retrieve column defaults with blk_default only during a bulk-copy-in operation. The application cannot call blk_default until blk_init(CS_BLK_IN) returns CS_SUCCEED.

See also

blk_bind, blk_describe, blk_init

blk_describe

Description

Retrieve a description of a database column.

Syntax

CS_RETCODE blk_describe(blkdesc, colnum, datafmt)

```
CS_BLKDESC    *blkdesc;
CS_INT        colnum;
CS_DATAFMT    *datafmt;
```

Parameters

blkdesc

A pointer to the CS_BLKDESC that is serving as a control block for the bulk-copy operation. blk_alloc allocates a CS_BLKDESC structure.

colnum

The number of the column of interest. The first column in a table is column number 1, the second is number 2, and so forth.

datafmt

A pointer to a CS_DATAFMT structure. `blk_describe` fills **datafmt* with a description of the database column referenced by *colnum*.

During a bulk-copy-in operation, `blk_describe` fills in the following fields in the CS_DATAFMT:

Table 4-4: CS_DATAFMT fields, as set by `blk_describe` for bulk copy in

Field name	<code>blk_describe</code> Sets the field to
<i>name</i>	The null-terminated name of the column, if any. A NULL name is indicated by a <i>namelen</i> of 0.
<i>namelen</i>	The actual length of the name, not including the null terminator. 0 to indicate a NULL <i>name</i> .
<i>datatype</i>	A type constant representing the datatype of the column. All type constants listed on the “Types” topics page are valid, with the exception of CS_VARCHAR_TYPE and CS_VARBINARY_TYPE.
<i>maxlength</i>	The maximum possible length of the data for the column.
<i>scale</i>	The scale of the column.
<i>precision</i>	The precision of the column.

During a bulk-copy-out operation, `blk_describe` fills in the following fields in the CS_DATAFMT:

Table 4-5: CS_DATAFMT fields, as set by `blk_describe` for bulk copy out

Field name	<code>blk_describe</code> Sets the field to
<i>name</i>	The null-terminated name of the column, if any. A NULL name is indicated by a <i>namelen</i> of 0.
<i>namelen</i>	The actual length of the name, not including the null terminator. 0 to indicate a NULL <i>name</i> .
<i>datatype</i>	The datatype of the column. All datatypes listed on the “Types” topics page in the <i>Open Client Client-Library/C Reference Manual</i> are valid.
<i>maxlength</i>	The maximum possible length of the data for the column.
<i>scale</i>	The scale of the column.
<i>precision</i>	The precision of the column.

Field name	blk_describe Sets the field to
<i>status</i>	A bit mask of the following symbols, combined with a bitwise OR: <ul style="list-style-type: none"> • CS_CANBENULL to indicate that the column can contain NULL values. • CS_HIDDEN to indicate that this column is a hidden column that has been exposed. Hidden columns are exposed when the CS_HIDDEN_KEYS property is set for the bulk descriptor's parent connection. • CS_IDENTITY to indicate that the column is an identity column. • CS_KEY to indicate the column is part of the key for a table. • CS_VERSION_KEY to indicate the column is part of the version key for the row.
<i>usertype</i>	The Adaptive Server user-defined datatype of the column, if any. <i>usertype</i> is set in addition to (not instead of) <i>datatype</i> .
<i>locale</i>	A pointer to a CS_LOCALE structure that contains locale information for the data.

Return value

blk_describe returns:

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

blk_describe returns CS_FAIL if *colnum* does not represent a valid result column.

Usage

- blk_describe is a client-side routine.
- blk_describe describes the format of a database column. The application can use this information to:
 - Determine the datatype and size requirements for allocating storage for retrieving rows (for bulk copy out of the database).
 - Determine compatibility between program variable datatypes and the database columns (by calling cs_will_convert to determine whether the conversion is supported and, if necessary, by checking the data lengths).
 - Perform error checking. For example, the debug version of a bulk-copy application might call blk_describe to confirm assumptions about the format of table columns.

- An application typically uses a column description while determining compatible program variable types and sizes.
- See the “CS_DATAFMT Structure” topics page in the *Open Client Client-Library/C Reference Manual* for a complete description of the CS_DATAFMT structure.

See also `blk_default`, `blk_init`

blk_done

Description Mark a complete bulk-copy operation or a complete bulk-copy batch.

Syntax `CS_RETCODE blk_done(blkdesc, type, outrow)`

```
CS_BLKDESC    *blkdesc;
CS_INT        type;
CS_INT        *outrow;
```

Parameters

blkdesc

A pointer to the CS_BLKDESC that is serving as a control block for the bulk-copy operation. `blk_alloc` allocates a CS_BLKDESC structure.

type

One of the following symbolic values:

Value of type	blk_done
CS_BLK_ALL	Marks a complete bulk-copy-in or bulk-copy-out operation.
CS_BLK_BATCH	Marks the end of a batch of rows in a batched bulk-copy- in operation.
CS_BLK_CANCEL	Cancels a bulk-copy batch or bulk-copy operation.

outrow

A pointer to an integer variable. If *type* is CS_BLK_BATCH or CS_BLK_ALL, `blk_done` sets **outrow* to the number of rows bulk copied to Adaptive Server since the application’s last `blk_done` call. When *type* is CS_BLK_CANCEL, **outrow* is set to 0.

Return value

`blk_done` returns:

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Returns	To indicate
CS_PENDING	Asynchronous network I/O is in effect. For more information, see the “Asynchronous Programming” topics page in the <i>Open Client Library/C Reference Manual</i> .

Common reasons for blk_done failure include:

- An invalid *blkdesc* pointer
- An invalid value for *type*

Examples

```
/*
** BulkCopyIn()
*/

CS_STATIC CS_RETCODE
BulkCopyIn(connection)
CS_CONNECTION *connection;
{
    CS_BLKDESC *blkdesc;
    CS_DATAFMT datafmt; /* variable descriptions */
    Blk_Data *dptr; /* data for transfer */
    CS_INT datalen[5]; /* variable data length */
    CS_INT len;
    CS_INT numrows;

    /*
    ** Ready to start the bulk copy in now that all the
    ** connections have been made and have a table name.
    ** Start by getting the bulk descriptor initializing.
    */
    ...CODE DELETED....

    /*

    ** Now to bind the variables to the columns and

    ** transfer the data
    */
    ...CODE DELETED....

    /* ALL the rows sent so clear up */
    if (blk_done(blkdesc, CS_BLK_ALL, &numrows) == CS_FAIL)
    {
```

```

        ex_error("BulkCopyIn: blk_done() failed");
        return CS_FAIL;
    }
    if (blk_drop(blkdesc) == CS_FAIL)
    {
        ex_error("BulkCopyIn: blk_drop() failed");
        return CS_FAIL;
    }
    return CS_SUCCEED;
}

```

Usage

- `blk_done` is a client-side routine. However, it is necessary in both client-only and gateway applications.
- Calling `blk_done` with `type` as `CS_BLK_ALL` marks the end of a bulk-copy operation. Once an application marks the end of a bulk-copy operation, it cannot call any Bulk-Library routines (except for `blk_drop` and `blk_alloc`) until it begins a new bulk-copy operation by calling `blk_init`.
- Calling `blk_done` with `type` as `CS_BLK_BATCH` marks the end of a batch of rows in a bulk-copy-in operation. `CS_BLK_BATCH` is legal only during bulk-copy-in operations.
- Calling `blk_done` with `type` as `CS_BLK_CANCEL` cancels the current bulk-copy operation. Rows transferred since an application's last `blk_done(CS_BLK_BATCH)` call are not saved in the database. Once an application cancels a bulk-copy operation, it cannot call any bulk-copy routines (except for `blk_drop` and `blk_alloc`) until it initializes a new bulk-copy operation by calling `blk_init`.

Calling `blk_done` during Bulk-Copy-In operations

- When an application bulk copies data into a database, the rows are permanently saved only when the application calls `blk_done`. During a large data transfer, `blk_done(CS_BLK_BATCH)` can be called periodically to “batch” the transmitted rows into smaller units of recoverability.
- An application can batch rows by calling `blk_done` with `type` as `CS_BLK_BATCH` once every n rows or when there is a lull between periods of data, as in a telemetry application. This causes all rows transferred since the application's last `blk_done` call to be permanently saved.
- After saving a batch of rows, an application's first call to `blk_rowxfer` or `blk_rowxfer_mult` implicitly starts the next batch.

- An application must call `blk_done` with type as `CS_BLK_ALL` to send its final batch of rows. This call permanently saves the rows, marks the end of the bulk-copy operation, and cleans up internal bulk-copy data structures.

Calling *blk_done* during Bulk-Copy-Out operations

- After transferring the last row in a bulk-copy-out operation, an application must call `blk_done` with type as `CS_BLK_ALL` to mark the end of the bulk-copy operation and clean up internal bulk-copy data structures.

See also

`blk_init`, `blk_rowxfer`, `blk_rowxfer_mult`

blk_drop

Description Deallocate a `CS_BLKDESC` structure.

Syntax `CS_RETCODE blk_drop(blkdesc)`

Parameters `blkdesc`
A pointer to a `CS_BLKDESC` previously allocated via `blk_alloc`.

Return value `blk_drop` returns:

Returns	To Indicate
<code>CS_SUCCEEDED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.

Examples

```
/*
** BulkCopyIn()
*/

CS_STATIC CS_RETCODE
BulkCopyIn(connection)
CS_CONNECTION *connection;
{
    CS_BLKDESC *blkdesc;
    CS_DATAFMT datafmt; /* variable descriptions */
    Blk_Data *dptr; /* data for transfer */
    CS_INT datalen[5]; /* variable data length */
    CS_INT len;
```



```

CS_INT      numrows;

/*
** Ready to start the bulk copy in now that all the
** connections have been made and have a table name.
** Start by getting the bulk descriptor initializing.
*/
...CODE DELETED.....

/*
** Now to bind the variables to the columns and
** transfer the data
*/
...CODE DELETED.....

/* ALL the rows sent so clear up */
if (blk_done(blkdesc, CS_BLK_ALL, &numrows) == CS_FAIL)
{
    ex_error("BulkCopyIn: blk_done() failed");
    return CS_FAIL;
}
if (blk_drop(blkdesc) == CS_FAIL)
{
    ex_error("BulkCopyIn: blk_drop() failed");
    return CS_FAIL;
}

return CS_SUCCEED;
}

```

- Usage
- A `CS_BLKDESC` structure, also called a *bulk-descriptor structure*, contains information about a particular bulk-copy operation.
 - Once a bulk-descriptor structure has been deallocated, it cannot be used again. To allocate a new `CS_BLKDESC`, an application can call `blk_alloc`.
 - `blk_drop` is typically called after `blk_done`. It must be the last routine called in a bulk-copy operation.
- See also `blk_alloc`, `blk_done`

blk_getrow

Description Server-side routine to retrieve and store a formatted bulk-copy row.

Syntax CS_RETCODE blk_getrow(srvproc, blkdescp, rowp)

```

SRV_PROC      *srvproc;
CS_BLKDESC   *blkdescp;
CS_BLK_ROW   *rowp;
    
```

Parameters

srvproc

A pointer to the SRV_PROC structure associated with the client sending the bulk-copy row. It contains all the information that Server-Library uses to manage communications and data between the Open Server and the client.

blkdescp

A pointer to a CS_BLKDESC structure containing information about bulk-copy data. This structure must have been previously allocated with a call to blk_alloc and initialized with a call to blk_init. This structure is used to interpret incoming formatted bulk-copy rows.

rowp

A pointer to a CS_BLK_ROW structure containing space for a formatted bulk-copy row. Space must have been previously allocated with blk_rowalloc.

The CS_BLK_ROW structure is a hidden structure that holds formatted bulk-copy rows sent from the client.

Return value

blk_getrow returns:

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_END_DATA	There are no more rows.
CS_BLK_HAS_TEXT	The row contains some text, image, sensitivity, or boundary data. Use blk_gettext to retrieve the text, image, sensitivity, or boundary data. Note that a return value of CS_BLK_HAS_TEXT implies a successful return, just like CS_SUCCEED.
CS_FAIL	The routine failed.

Usage

- blk_getrow is a server-side routine that is useful in gateway applications.
- This routine copies the incoming formatted bulk-copy row into the CS_BLK_ROW structure to which rowp points. The row data is saved only until the next call to blk_getrow. The application must have previously allocated the space for the row using blk_rowalloc.

- Once a row has been received via `blk_getrow`, the application may examine the contents of any fields (other than text, image, sensitivity, or boundary fields) using `blk_colval`.
- Use `blk_gettext` to retrieve text, image, sensitivity, and boundary fields.
- A bulk-copy row may subsequently be sent to another server using the `blk_sendrow` routine.
- An application must read all incoming rows with `blk_getrow`, until there are no more rows.
- Once `blk_getrow` returns `CS_END_DATA`, the application must drop the space allocated for the row using `blk_rowdrop`.

See also

`blk_colval`, `blk_gettext`, `blk_rowalloc`

blk_gettext

Description

Server-side routine to retrieve the text, image, sensitivity, or boundary portion of an incoming formatted bulk-copy row.

Syntax

```
CS_RETCODE blk_gettext(srvproc,blkdesc, rowp, bufp, bufsize, outlenp)
    SRV_PROC      *srvproc;
    CS_BLKDESC    *blkdesc;
    CS_BLK_ROW    *rowp;
    CS_BYTE       *bufp;
    CS_INT        bufsize;
    CS_INT        *outlenp;
```

Parameters

srvproc

A pointer to the `SRV_PROC` structure associated with the client sending the bulk-copy row. This structure contains all the information that Server-Library uses to manage communications and data between the Open Server application and the client.

blkdesc

A pointer to a `CS_BLKDESC` structure containing information about bulk-copy data. This structure must have been previously allocated with a call to `blk_alloc` and initialized with a call to `blk_init`. This structure is used to interpret incoming formatted bulk-copy rows.

rowp

A pointer to the formatted bulk-copy row read from the client via a prior call to blk_getrow.

The CS_BLK_ROW structure is a hidden structure that holds formatted bulk-copy rows sent from the client.

bufp

A pointer to the application buffer in which Bulk-Library places the text, image, sensitivity, or boundary data.

bufsize

The size, in bytes, of the space pointed at by *bufp*.

outlenp

A pointer to a CS_INT variable, which is set to the number of bytes actually read by blk_gettext. It may be less than *bufsize*. To determine whether all of the text, image, sensitivity, or boundary part of the row has been read, check for a return code of CS_END_DATA. A *outlenp value that is less than *bufsize* does not necessarily indicate the end of a row. For example, it could indicate the end of a text, image, sensitivity, or boundary column that is not the last column in the row.

Return value

blk_gettext returns:

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_END_DATA	There are no more text, image, sensitivity, or boundary fields for the current incoming bulk-copy row. Call blk_getrow to get the next bulk-copy row.
CS_FAIL	The routine failed.

Usage

- blk_gettext is a server-side routine that is useful in gateway applications.
- This routine is used with blk_getrow and blk_colval to receive formatted bulk-copy rows and route them to an Adaptive Server. This routine retrieves the text, image, sensitivity, or boundary portions of the row.

- Bulk-copy rows are formatted so that all text, image, sensitivity, and boundary fields occur at the end of the row, after all the other types of fields. To route a row to an Adaptive Server, first call `blk_getrow` to retrieve all the parts of the row containing other types of fields. Then call `blk_colval` to retrieve and store portions of the row containing other types of fields. Decide where this data goes and then send it to the remote server, using `blk_sendrow`. Then, call `blk_gettext` to copy text, image, sensitivity, or boundary data into an application buffer. Finally, call `blk_sendtext` to send this information to the remote server.
- If an incoming bulk-copy row has any text, image, sensitivity, or boundary fields, `blk_getrow` returns `CS_BLK_HAS_TEXT`.
- It is not an error to call `blk_gettext` if the row contains no text, image, sensitivity, or boundary fields. The routine simply returns `CS_END_DATA`.
- This routine must be called after `blk_getrow`. Also, it must be called until it returns `CS_END_DATA`, to fully read in a bulk-copy row.
- Before rows can be sent to a server, the gateway application must have set up the bulk-copy operation with a call to `blk_init`.
- It is critical that the table for which the bulk-copy operation was initialized and the table into which the client is bulk copying are the same table.

See also

`blk_colval`, `blk_getrow`, `blk_gettext`, `blk_sendtext`

blk_init

Description

Initiate a bulk-copy operation.

Syntax

```
CS_RETCODE blk_init(blkdesc, direction, tablename,
                    tnamelen)
```

```
CS_BLKDESC  *blkdesc;
CS_INT      direction;
CS_CHAR      *tablename;
CS_INT      tnamelen;
```

Parameters

blkdesc

A pointer to the CS_BLKDESC controlling the bulk-copy operation. An application can allocate a CS_BLKDESC by calling blk_alloc.

The parent connection of the CS_BLKDESC must be open when blk_init is called and cannot have any pending results.

direction

One of the following symbolic values, to indicate the direction of the bulk-copy operation:

Value of direction	blk_init
CS_BLK_IN	Begins a bulk-copy operation to upload rows from the client to the server.
CS_BLK_OUT	Begins a bulk-copy operation to download rows from the server to the client.

tablename

A pointer to the name of the table of interest. Any legal server table name is acceptable.

tnamelen

The length, in bytes, of **tablename*. If **tablename* is null-terminated, pass *tnamelen* as CS_NULLTERM.

Return value

blk_init returns:

Returns	To indicate
CS_SUCCEEDED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_PENDING	Asynchronous network I/O is in effect. For more information, see the “Asynchronous Programming” topics page in the <i>Open Client Library/C Reference Manual</i> .

A common cause of failure is specifying a non-existent table.

Examples

```

/*
** BulkCopyIn()
** Ex_tabname is globally defined.
*/
CS_STATIC CS_RETCODE
BulkCopyIn(connection)
CS_CONNECTION *connection;

```

```

{
    CS_BLKDESC *blkdesc;
    CS_DATAFMT datafmt; /* variable descriptions */
    Blk_Data *dptr; /* data for transfer */
    CS_INT datalen[5]; /* variable data length */
    CS_INT len;
    CS_INT numRows;

    /*
    ** Ready to start the bulk copy in now that all the
    ** connections have been made and have a table name.
    ** Start by getting the bulk descriptor and
    ** initializing.
    */
    if (blk_alloc(connection, BLK_VERSION_100, &blkdesc)
        != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_alloc() failed");
        return CS_FAIL;
    }

    if (blk_init(blkdesc, CS_BLK_IN,
                Ex_tabname, strlen(Ex_tabname)) == CS_FAIL)
    {
        ex_error("BulkCopyIn: blk_init() failed");
        return CS_FAIL;
    }

    /*
    ** Bind the variables to the columns and send the rows,
    ** and then clean up.
    */
    ...CODE DELETED....

    return CS_SUCCEED;
}

```

Usage

- `blk_init` begins a bulk-copy operation.
- `blk_init` is a client-side routine. However, it is necessary in both client-only and gateway applications.
- Multiple `CS_BLKDESC` and `CS_COMMAND` structures can exist on the same connection, but only one `CS_BLKDESC` or `CS_COMMAND` structure can be active at the same time.
 - A bulk-copy operation begun with `blk_init` must be completed before the connection can be used for any other operation.

- A bulk-copy operation cannot be started when the connection is being used to initiate, send, or process the results of other Client-Library or Bulk-Library commands.
- When a bulk-copy operation is complete, an application must call `blk_done` with *type* as `CS_BLK_ALL` to mark the end of the bulk-copy operation and clean up internal Bulk-Library data structures.

See also

`blk_alloc`, `blk_bind`, `blk_done`, `blk_rowxfer_mult`

blk_props

Description

Set or retrieve bulk-descriptor structure properties.

Syntax

```
CS_RETCODE blk_props(blkdesc, action, property,
                    buffer, buflen, outlen)
```

```
CS_BLKDESC    *blkdesc;
CS_INT        action;
CS_INT        property;
CS_VOID       *buffer;
CS_INT        buflen;
CS_INT        *outlen;
```

Parameters

blkdesc

A pointer to a `CS_BLKDESC` structure. A bulk-descriptor structure contains information about a bulk-copy operation. `blk_alloc` allocates a bulk-descriptor structure.

action

One of the following symbolic constants:

Value of action	blk_props
<code>CS_SET</code>	Sets the value of the property.
<code>CS_GET</code>	Retrieves the value of the property.
<code>CS_CLEAR</code>	Clears the value of the property by resetting it to its default value.

property

A symbolic constant that indicates the property of interest. Table 4-6 on page 136 lists valid *property* constants and describes each property.

buffer

If a property value is being set, *buffer* points to the value to use in setting the property.

If a property value is being retrieved, *buffer* points to the space in which `blk_props` will place the requested information.

The C datatype of the value depends on the property. Refer to Table 4-6 on page 136 for the datatype of the property of interest.

buflen

Generally, *buflen* is the length, in bytes, of **buffer*.

If a property value is being set and the value in **buffer* is null-terminated, pass *buflen* as `CS_NULLTERM`.

If **buffer* is a fixed-length or symbolic value, pass *buflen* as `CS_UNUSED`.

outlen

A pointer to an integer variable.

If a property value is being set, *outlen* is not used and should be passed as `NULL`.

If a property value is being retrieved and *outlen* is supplied, `blk_props` sets **outlen* to the length, in bytes, of the requested information.

If the information is larger than *buflen* bytes, an application can use the value of **outlen* to determine how many bytes are needed to hold the information.

Return value

`blk_props` returns:

Returns	To indicate
<code>CS_SUCCEED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.

Usage

- Bulk-descriptor properties define aspects of a specific bulk-copy operation.
- Applications that set Bulk-Library properties must do so after calling `blk_alloc` to allocate a bulk-descriptor structure and before calling `blk_init` to initiate a specific bulk-copy operation.
- An application can use `blk_props` to set or retrieve the following properties:

Table 4-6: Client/Server bulk descriptor properties

Property name	Description	*buffer is	Applies to	Notes
BLK_IDENTITY	Whether a table's identity column is included in the bulk-copy operation.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	IN copies only	
BLK_NOAPI_CHK	Whether parameter and error checking for illegal parameter values and state transitions are disabled for Bulk-Library calls.	CS_TRUE or CS_FALSE. The default is CS_FALSE, which means error checking is performed.	Both IN and OUT copies	
BLK_SENSITIVITY_LBL	Whether a table's <i>sensitivity</i> column is included in the bulk-copy operation.	CS_TRUE or CS_FALSE. The default is CS_FALSE.	Both IN and OUT copies	Secure SQLServer only

BLK_IDENTITY property

- BLK_IDENTITY determines whether a table's identity column is included in a bulk-copy-in operation.
- BLK_IDENTITY does not affect bulk-copy-out operations.
- If BLK_IDENTITY is CS_TRUE, the application must supply data for the identity column.

If BLK_IDENTITY is CS_FALSE, the application does not need to supply data for the identity column. In this case, the server supplies a default value for the column.

- BLK_IDENTITY works by setting `identity_insert` on for the database table of interest. This allows values to be inserted into the identity column. When the bulk-copy operation is finished, the `identity_insert` option for the table is turned off.

For more information about `identity_insert`, see the *Adaptive Server Enterprise Reference Manual*.

BLK_NOAPI_CHK property •BLK_NOAPI_CHK can be set to CS_TRUE to disable parameter and state checking of Bulk-Library calls. The default is CS_FALSE, which enables parameter checking and state checking of each Bulk-Library call. These two types of error checking are described below:

- *Parameter checking* determines whether the application has passed valid parameters and combinations of parameters in the call.
- *State checking* ensures that calls are made in the required sequence. For example, `blk_init` must be called before `blk_bind`.

The default error checking ensures that your application calls Bulk-Library routines in the appropriate manner. With API checking enabled, a descriptive error message is raised when the application commits a usage error, and the routine that discovers the error returns `CS_FAIL`.

Warning! With API checking disabled, Bulk-Library usage errors may lead to unexpected behavior or even program crashes.

- If your application has been fully tested and completely debugged, you may see improved performance with API checking disabled. Bulk-Library also calls Client-Library internally, so to get the full benefit, you should also disable API checking in Client-Library (by calling `ct_config` to set the `CS_NOAPI_CHK` context property to `CS_TRUE`).
- `BLK_NOAPI_CHK` does not affect testing for errors, such as network errors or conversion overflow, that can occur in well-behaved applications.

BLK_SENSITIVITY_LBL property

- `BLK_SENSITIVITY_LBL` is useful in applications that perform bulk-copy operations to or from Secure SQL Server.
- `BLK_SENSITIVITY_LBL` determines whether or not data for the *sensitivity* column is included in a bulk-copy operation. By default, *sensitivity* column data is not included.
- `BLK_SENSITIVITY_LBL` affects both bulk-copy-in and bulk-copy-out operations.
- If `BLK_SENSITIVITY_LBL` is `CS_TRUE`, the application must supply data for the *sensitivity* column on bulk-copy-in operations and will receive data from the *sensitivity* column on bulk-copy-out operations.

If `BLK_SENSITIVITY_LBL` is `CS_FALSE`, the application does not need to supply data for the *sensitivity* column on bulk-copy-in operations and will not receive data from the *sensitivity* column on bulk-copy-out operations.

- BLK_SENSITIVITY_LBL is applicable to Secure SQL Server copies only. blk_init fails if BLK_SENSITIVITY_LBL is CS_TRUE and the application attempts a bulk-copy operation against a standard Adaptive Server.
- Application users copying into the *sensitivity* column must have the bcpin_labels_role role activated on Secure SQL Server. blk_init fails if the bcpin_labels_role is not activated for the connection's user.
- For more information about Secure SQL Server, see your Secure SQL Server documentation.

See also blk_alloc, blk_init

blk_rowalloc

Description Server-side routine to allocate space for a formatted bulk-copy row.

Syntax CS_RETCODE blk_rowalloc(srvproc, row)

```
SRV_PROC      *srvproc;  
CS_BLK_ROW    **row;
```

Parameters

srvproc

A pointer to the SRV_PROC structure associated with the client sending formatted bulk-copy rows. It contains all the information that Server-Library uses to manage communications and data between the Open Server and the client.

row

A pointer to a pointer to a CS_BLK_ROW structure.

The CS_BLK_ROW structure is a hidden structure that holds formatted bulk-copy rows sent from the client.

Return value

blk_rowalloc returns:

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.

Usage

- blk_rowalloc is a server-side routine that is useful in gateway applications.
- This routine allocates space in which blk_getrow will place the formatted bulk-copy row.

- The row space is used by all calls to `blk_getrow`.
- When all rows have been retrieved and sent to the remote server, call `blk_rowdrop` to drop the space allocated for the row.

See also `blk_getrow`, `blk_rowdrop`, `blk_gettext`

blk_rowdrop

Description Server-side routine to free space previously allocated for a formatted bulk-copy row.

Syntax `CS_RETCODE blk_rowdrop(srvproc, row)`

```
SRV_PROC      *srvproc;
CS_BLK_ROW    *row;
```

Parameters

srvproc

A pointer to the `SRV_PROC` structure associated with the client sending formatted bulk-copy rows. It contains all the information that Server-Library uses to manage communications and data between the Open Server application and the client.

row

A pointer to a hidden `CS_BLK_ROW` structure that was allocated by a call to `blk_rowalloc`.

Return value `blk_rowdrop` returns:

Returns	To indicate
<code>CS_SUCCEED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.

Usage

- `blk_rowdrop` is a server-side routine that is useful in gateway applications.
- This routine frees space previously allocated by `blk_rowalloc`.
- It must be called after all formatted bulk-copy rows have been retrieved and sent to the remote server.

See also `blk_getrow`, `blk_rowalloc`, `blk_gettext`

blk_rowxfer

Description Transfer one or more rows during a bulk-copy operation without specifying or receiving a row count.

Syntax CS_RETCODE blk_rowxfer(blkdesc)

CS_BLKDESC *blkdesc;

Parameters *blkdesc*

A pointer to the CS_BLKDESC that is serving as a control block for the bulk-copy operation. blk_alloc allocates a CS_BLKDESC structure.

Return value blk_rowxfer returns:

Table 4-7: blk_rowxfer return values

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_PENDING	Asynchronous network I/O is in effect. For more information, see the “Asynchronous Programming” topics page in the <i>Open Client Library/C Reference Manual</i> .
CS_BLK_HAS_TEXT	The row contains one or more columns which have been marked for transfer via blk_textxfer. The application must call blk_textxfer to transfer data for these columns before calling blk_rowxfer to transfer the next row.
CS_END_DATA	When copying data out from a database, blk_rowxfer returns CS_END_DATA to indicate that all rows have been transferred. When copying data into a database, blk_rowxfer does not return CS_END_DATA.
CS_ROW_FAIL	A recoverable error occurred while fetching a row. Applies to bulk-copy-out operations only. Recoverable errors include memory allocation failures and conversion errors (such as overflowing the destination buffer) that occur while copying row values to program variables. In the case of buffer-overflow errors, blk_rowxfer sets the corresponding <i>*indicator</i> variable(s) to a value greater than 0. Indicator variables must have been specified in the application’s calls to blk_bind. When blk_rowxfer returns CS_ROW_FAIL, the application must continue calling blk_rowxfer to keep retrieving rows, or it can call ct_cancel to cancel the remaining results.

Examples

```

/*
** BulkCopyIn()
** BLKDATA and DATA_END are defined in the bulk copy
** example program.
**
*/

CS_STATIC CS_RETCODE
BulkCopyIn(connection)
CS_CONNECTION *connection;
{
    CS_BLKDESC *blkdesc;
    CS_DATAFMT datafmt; /* variable descriptions */
    Blk_Data *dptr; /* data for transfer */
    CS_INT datalen[5]; /* variable data length */
    CS_INT len;
    CS_INT numrows;

/*
** Ready to start the bulk copy in now that all the
** connections have been made and have a table name.
** Start by getting the bulk descriptor initializing.
**
*/
    ...CODE DELETED.....

/*
** Now to bind the variables to the columns and
** transfer the data
**
*/
    datafmt.locale = 0;
    datafmt.count = 1;
    dptr = BLKDATA;
    while (dptr->pub_id != DATA_END)
    {
        datafmt.datatype = CS_INT_TYPE;
        datafmt.maxlength = sizeof(CS_INT);
        datalen[0] = CS_UNUSED;

        if (blk_bind(blkdesc, 1, &datafmt, &dptr->pub_id,
            &datalen[0], NULL) != CS_SUCCEEDED)
        {
            ex_error("BulkCopyIn: blk_bind(1) failed");
            return CS_FAIL;
        }
        datafmt.datatype = CS_CHAR_TYPE;
        datafmt.maxlength = MAX_PUBNAME - 1;
    }
}

```

```
    datalen[1] = strlen(dptr->pub_name);
    if (blk_bind(blkdesc, 2, &datafmt, dptr->pub_name,
                &datalen[1], NULL) != CS_SUCCEEDED)
    {
        ex_error("BulkCopyIn: blk_bind(2) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_PUBCITY - 1;
    datalen[2] = strlen(dptr->pub_city);
    if (blk_bind(blkdesc, 3, &datafmt, dptr->pub_city,
                &datalen[2], NULL) != CS_SUCCEEDED)
    {
        ex_error("BulkCopyIn: blk_bind(3) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_PUBST - 1;
    datalen[3] = strlen(dptr->pub_st);
    if (blk_bind(blkdesc, 4, &datafmt, dptr->pub_st,
                &datalen[3], NULL) != CS_SUCCEEDED)
    {
        ex_error("BulkCopyIn: blk_bind(4) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_BIO - 1;
    datalen[4] = strlen((char *)dptr->pub_bio);
    if (blk_bind(blkdesc, 5, &datafmt, dptr->pub_bio,
                &datalen[4], NULL) != CS_SUCCEEDED)
    {
        ex_error("BulkCopyIn: blk_bind(5) failed");
        return CS_FAIL;
    }
    if (blk_rowxfer (blkdesc) == CS_FAIL)
    {
        ex_error("BulkCopyIn: blk_rowxfer() failed");
        return CS_FAIL;
    }
    dptr++;
}

/* ALL the rows sent so clear up */
...CODE DELETED.....

return CS_SUCCEEDED;
}
```

Usage

- blk_rowxfer is a client-side routine.

- `blk_rowxfer` is equivalent to calling `blk_rowxfer_mult` with a `NULL` `row_count` parameter.
- See `blk_rowxfer_mult` in this chapter for more information.

See also `blk_bind`, `blk_rowxfer_mult`, `blk_textxfer`

blk_rowxfer_mult

Description Transfer one or more rows during a bulk-copy operation.

Syntax `CS_RETCODE blk_rowxfer_mult(blkdesc, row_count)`

```
CS_BLKDESC    *blkdesc;
CS_INT        *row_count;
```

Parameters

blkdesc

A pointer to the `CS_BLKDESC` that is serving as a control block for the bulk-copy operation. `blk_alloc` allocates a `CS_BLKDESC` structure.

row_count

A pointer to a `CS_INT` variable or `NULL`.

For bulk-copy-out operations, `blk_rowxfer_mult` returns with **row_count* set to the number of rows read by the call. If *row_count* is `NULL`, this information is not available to the application. (The application can call `blk_done` to determine how many rows have been transferred by the cumulative number of `blk_rowxfer_mult` calls since the last `blk_done` call—but it is simpler to use a row count variable.

For bulk-copy-in operations, `blk_rowxfer_mult` sends the number of rows specified by **row_count* to the server. If *row_count* is `NULL` or **row_count* is 0, then the number of rows specified by *datafmt->count* in previous calls to `blk_bind` are sent to the server.

row_count is used by applications that perform array binding. For more information on this feature, see “Array binding” on page 116.

Return value `blk_rowxfer_mult` returns:

Table 4-8: *blk_rowxfer_mult* return values

Returns	To indicate
<code>CS_SUCCEED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.

Returns	To indicate
CS_PENDING	Asynchronous network I/O is in effect. For more information, see the “Asynchronous Programming” topics page in the <i>Open Client Library/C Reference Manual</i> .
CS_BLK_HAS_TEXT	The row contains one or more columns which have been marked for transfer via blk_textxfer. The application must call blk_textxfer to transfer data for these columns row before calling blk_rowxfer_mult to transfer the next row.
CS_END_DATA	When copying data out from a database, blk_rowxfer_mult returns CS_END_DATA to indicate that all rows have been transferred. When copying data into a database, blk_rowxfer_mult does not return CS_END_DATA.
CS_ROW_FAIL	A recoverable error occurred while fetching a row. Applies to bulk-copy-out operations only. blk_rowxfer_mult sets *row_count to indicate the number of rows transferred (including the row containing the error) and transfers no rows after that row. The next call to blk_rowxfer_mult will read rows starting with the row after the one where the error occurred. Recoverable errors include memory allocation failures and conversion errors (such as overflowing the destination buffer) that occur while copying row values to program variables. In the case of buffer-overflow errors, blk_rowxfer_mult sets the corresponding *indicator variable(s) to a value greater than 0. Indicator variables must have been specified in the application’s calls to blk_bind. When blk_rowxfer_mult returns CS_ROW_FAIL, the application must continue calling blk_rowxfer_mult to keep retrieving rows, or it can call ct_cancel to cancel the remaining results.

A common reason for a blk_rowxfer_mult failure is conversion error.

Usage

- blk_rowxfer_mult is a client-side routine.
- An application calls blk_rowxfer_mult to transfer rows between program variables (bound with blk_bind) and the database table:
 - During a bulk-copy-in operation, blk_rowxfer_mult copies data from program variables to the database.

- During a bulk-copy-out operation, `blk_rowxfer_mult` copies data from the database and places it in program variables.
- Application variables must first be bound to table columns with `blk_bind` for `blk_rowxfer_mult` to read or write their contents.

blk_rowxfer_mult and Bulk-Copy-In operations

- To transfer rows into a database, an application calls `blk_rowxfer_mult` repeatedly to transfer values from program variables to the database table. See “Program structure for Bulk-Copy-In operations” on page 95 for the sequence of Bulk-Library calls used to transfer data into a database table.
- During bulk-copy-in operations, the value of `blk_rowxfer_mult`’s `*row_count` parameter overrides the array lengths that were passed to `blk_bind` (as `datafmt->count`). The number of rows transferred per call is determined as follows:
 - If the application passes the address of a row count variable as the `row_count` parameter, then `blk_rowxfer_mult` transfers either `datafmt->count` or `*row_count` rows, whichever is smaller.
 - If the application passes `row_count` as NULL, `blk_rowxfer_mult` always transfers `datafmt->count` rows.

For example, if an application was uploading 103 rows and it used array binding to transfer 10 rows at a time, the application would:

- Pass `datafmt->count` as 10 in all calls to `blk_bind`
- Set `*row_count` to 10 for the first 10 calls to `blk_rowxfer_mult`
- Set `*row_count` to 3 for the final call to `blk_rowxfer_mult`
- To upload row data that contains large text or image column values, you can forgo array binding and use `blk_textxfer` together with `blk_rowxfer_mult` to send large values one piece at a time. See “Transferring large text or image values in chunks” on page 146 for details.
- A bulk-copy-in operation is not automatically terminated if `blk_rowxfer_mult` returns CS_FAIL. An application can continue to call `blk_rowxfer_mult` after correcting or discarding the problem row.

blk_rowxfer_mult and Bulk-Copy-Out operations

- To transfer rows out of a database, an application calls `blk_rowxfer_mult` repeatedly to read column values from the server and place them in program variables. See “Program structure for Bulk-Copy-Out operations” on page 97 for the sequence of Bulk-Library calls used to read data from a database table.

- For bulk copies out of a database, the use of blk_rowxfer_mult is similar to the use of the Client-Library ct_fetch routine.
- The number of rows to be read by blk_rowxfer_mult is determined by the value passed as *datafmt*→*count* in the application's calls to blk_bind. blk_rowxfer_mult attempts to read this number of rows and write the data to program variables.

Fewer rows may be read by the final call to blk_rowxfer_mult (that is, the call that retrieves the last row in the table) or if a conversion error occurs while data is being retrieved. The former condition is indicated by a return code of CS_END_DATA; the latter, by CS_ROW_FAIL. In either case, blk_rowxfer_mult returns with **row_count* set to the actual number of rows read.

- To download row data that contains large text or image column values, you can forgo array binding and use blk_textxfer together with blk_rowxfer_mult to read large values one piece at a time. See Transferring large text or image values in chunks, below, for details.

Transferring large *text* or *image* values in chunks

- If array binding is not in effect, an application can use blk_textxfer in conjunction with blk_rowxfer_mult to transfer rows containing large text or image values. For information on how to do this, see “Bulk-Library client programming” on page 91.
- For tables that contain large text or image columns, it is often convenient for an application to transfer the text or image data in fixed-size chunks rather than all at once. If a column is transferred all at once, the application must have sufficient buffer space to hold the value in its entirety.
- To transfer large column values in chunks:
 - The application passes *buffer* as NULL in its blk_bind call for the column. This setting specifies that data for this column will be transferred using blk_textxfer. For a bulk-copy-in operation, the application must also specify the length of the column value as blk_bind's **datalen* parameter.
 - The application calls blk_rowxfer_mult to transfer the row. blk_rowxfer_mult returns CS_BLK_HAS_TEXT, indicating that Bulk-Library expects further data for this row to be transferred with blk_textxfer.
 - For each column requiring transfer, the application calls blk_textxfer in a loop until blk_textxfer returns CS_END_DATA, indicating that all of the data for this column has been transferred.

See also `blk_bind`, `blk_textxfer`

blk_sendrow

Description Server-side routine to send a formatted bulk-copy row obtained from `blk_getrow`.

Syntax `CS_RETCODE blk_sendrow(blkdesc, row)`

```
CS_BLKDESC  *blkdesc;
CS_BLK_ROW  *row;
```

Parameters

blkdesc

A pointer to the `CS_BLKDESC` that is serving as a control block for the bulk-copy operation. `blk_alloc` allocates a `CS_BLKDESC` structure.

row

A pointer to a `CS_BLK_ROW` structure. The `CS_BLK_ROW` is a hidden structure that holds formatted bulk-copy rows sent from the client. A gateway application can fill in a `CS_BLK_ROW` structure with a formatted row by calling the server-side routine `blk_getrow`.

Return value

`blk_sendrow` returns:

Table 4-9: blk_sendrow return values

Returns	To indicate
<code>CS_SUCCEED</code>	The routine completed successfully.
<code>CS_FAIL</code>	The routine failed.
<code>CS_BLK_HAS_TEXT</code>	The row contains one or more text, image, sensitivity, or boundary columns. The application must call <code>blk_gettext</code> and <code>blk_sendtext</code> to transfer the columns for this row before calling <code>blk_getrow</code> and <code>blk_sendrow</code> to transfer the next row.
<code>CS_PENDING</code>	Asynchronous network I/O is in effect. For more information, see the “Asynchronous Programming” topics page in the <i>Open Client Client-Library/C Reference Manual</i> .

Usage

- `blk_sendrow` is a server-side routine.
- A gateway application uses `blk_sendrow` in conjunction with `blk_getrow`. Together, the two routines enable a gateway application to receive formatted bulk-copy rows from an Open Client application and send them on to Adaptive Server.

- blk_sendrow is a gateway-specific substitute for blk_rowxfer or blk_rowxfer_mult. An application can call blk_sendrow only after calling blk_getrow to retrieve a formatted row.
- The sequence of calls in the gateway application is:
 - blk_getrow, to obtain a formatted bulk-copy row
 - blk_sendrow, to send the formatted row to Adaptive Server

If blk_getrow returns CS_BLK_HAS_TEXT, the application must call the following routines in a loop, until blk_gettext returns CS_END_DATA:

- blk_gettext, to pick up a chunk of text, image, sensitivity, or boundary data
- blk_sendtext, to send a chunk of text, image, sensitivity, or boundary data

Only one blk_gettext/blk_sendtext loop is required, no matter how many text, image, sensitivity, or boundary columns are being transferred.

See also

blk_init, blk_sendtext, blk_colval, blk_getrow, blk_gettext

blk_sendtext

Description Server-side routine to send text, image, sensitivity, or boundary data in a formatted bulk-copy row obtained from blk_getrow.

Syntax CS_RETCODE blk_sendtext(blkdesc, row, buffer, buflen)

```
CS_BLKDESC *blkdesc;  
CS_BLK_ROW *row;  
CS_BYTE *buffer;  
CS_INT buflen;
```

Parameters

blkdesc

A pointer to the CS_BLKDESC that is serving as a control block for the bulk-copy operation. blk_alloc allocates a CS_BLKDESC structure.

row

A pointer to a CS_BLK_ROW structure. The CS_BLK_ROW structure is a hidden structure that holds formatted bulk-copy rows sent from the client. A gateway application can fill in a CS_BLK_ROW structure with a formatted row by calling the routine blk_getrow.

buffer

A pointer to the space from which `blk_sendtext` picks up the chunk of text, image, sensitivity, or boundary data.

buflen

The length, in bytes, of the **buffer* data space.

Return value

`blk_sendtext` returns:

Table 4-10: *blk_sendtext* return values

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_PENDING	Asynchronous network I/O is in effect. For more information, see the “Asynchronous Programming” topics page in the <i>Open Client Client-Library/C Reference Manual</i> .

Usage

- `blk_sendtext` is a client-side routine.
- A gateway application uses `blk_sendtext` in conjunction with `blk_gettext`. Together, the two routines enable a gateway application to receive chunks of text, image, sensitivity, or boundary data in formatted bulk-copy rows from an Open Client application and send them on to Adaptive Server.
- `blk_sendtext` is a gateway-specific substitute for `blk_textxfer`. An application can call `blk_sendtext` only after calling `blk_gettext` to retrieve a chunk of text, image, sensitivity, or boundary data belonging to a formatted row.

- The sequence of calls in the gateway application is:

- `blk_getrow`, to pick up a formatted bulk-copy row
- `blk_sendrow`, to send the formatted row to Adaptive Server

If `blk_sendrow` returns `CS_BLK_HAS_TEXT`, the application must call the following routines in a loop, until `blk_gettext` returns `CS_END_DATA`:

- `blk_gettext`, to pick up a chunk of text, image, sensitivity, or boundary data
- `blk_sendtext`, to send a chunk of text, image, sensitivity, or boundary data

Only one `blk_gettext/blk_sendtext` loop is required, no matter how many text, image, sensitivity, or boundary columns are being transferred.

See also

`blk_init`, `blk_sendrow`, `blk_colval`, `blk_getrow`, `blk_gettext`

blk_srvinit

Description Server-side routine to copy descriptions of server table columns to the client, if required.

Syntax CS_RETCODE blk_srvinit(srvproc, blkdescp)

```
SRV_PROC      *srvproc;  
CS_BLKDESC   *blkdescp;
```

Parameters

srvproc

A pointer to the SRV_PROC structure associated with the client receiving column descriptions. It contains all the information that Server-Library uses to manage communications and data between the Open Server application and the client.

blkdescp

A pointer to a structure containing information about bulk-copy data. This structure must have been previously allocated with a call to blk_alloc and initialized via a call to blk_init. This structure is used to correctly interpret incoming formatted bulk-copy rows.

Return value blk_srvinit returns:

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed; no action was taken.

Usage

- blk_srvinit is a server-side routine that is useful in gateway applications.
- This routine sends the current server table column descriptions in the CS_BLKDESC structure to the client, if the client's TDS (Tabular Data Stream™) version is 5.0 or later.
- This routine must be called from within a SRV_LANGUAGE event handler, in response to an "insert bulk" request from the client.
- Once blk_srvinit has successfully returned descriptions to the client, the Open Server application's SRV_BULK event handler can begin reading bulk data from the client. The event handler first calls blk_rowalloc, then calls blk_getrow and blk_sendrow in a loop to transfer the bulk-copy rows.
- blk_init places the descriptions in the CS_BLKDESC structure, so the gateway application must call blk_init before calling blk_srvinit.

See also

blk_init, blk_getrow, blk_rowalloc, blk_sendrow

blk_textxfer

Description	Transfer a column's data in chunks during a bulk-copy operation.
Syntax	CS_RETCODE blk_textxfer(blkdesc, buffer, buflen, outlen)
	<pre> CS_BLKDESC *blkdesc; CS_BYTE *buffer; CS_INT buflen; CS_INT *outlen; </pre>
Parameters	<p><i>blkdesc</i> A pointer to the CS_BLKDESC that is serving as a control block for the bulk-copy operation. blk_alloc allocates a CS_BLKDESC structure.</p> <p><i>buffer</i> A pointer to the space from which blk_textxfer picks up the chunk of text, image, sensitivity, or boundary data.</p> <p><i>buflen</i> The length, in bytes, of the <i>buffer</i> data space.</p> <p><i>outlen</i> A pointer to an integer variable.</p> <p><i>outlen</i> is not used for a bulk-copy-in operation and should be passed as NULL.</p> <p>For a bulk-copy-out operation, <i>*outlen</i> represents the length, in bytes, of the data copied to <i>*buffer</i>.</p>
Return value	blk_textxfer returns:

Table 4-11: blk_textxfer return values

Returns	To indicate
CS_SUCCEED	The routine completed successfully.
CS_FAIL	The routine failed.
CS_END_DATA	When copying data out from a database, blk_textxfer returns CS_END_DATA to indicate that a complete column value has been sent. When copying data into a database, blk_textxfer returns CS_END_DATA when an amount of data equal to blk_bind's <i>*datalen</i> has been sent.
CS_PENDING	Asynchronous network I/O is in effect. For more information, see the "Asynchronous Programming" topics page in the <i>Open Client Client-Library/C Reference Manual</i> .

Examples

```
/*
** BulkCopyIn()
**
** BLKDATA and DATA_END are defined in the bulk copy
** example program.
**/

CS_STATIC CS_RETCODE
BulkCopyIn(connection)
CS_CONNECTION *connection;
{
    CS_BLKDESC *blkdesc;
    CS_DATAFMT datafmt; /* variable descriptions */
    Blk_Data *dptr; /* data for transfer */
    CS_INT datalen[5]; /* variable data length */
    CS_INT len;
    CS_INT numrows;

    /*
    ** Ready to start the bulk copy in now that all the
    ** connections have been made and have a table name.
    ** Start by getting the bulk descriptor initializing.
    **/
    ...CODE DELETED....

    /* Bind columns and transfer rows */
    dptr = BLKDATA;
    while (dptr->pub_id != DATA_END)
    {
        datafmt.datatype = CS_INT_TYPE;
        datafmt.count = 1;
        datafmt.maxlength = sizeof(CS_INT);
        datalen[0] = CS_UNUSED;

        if (blk_bind(blkdesc, 1, &datafmt, &dptr->pub_id,
                    &datalen[0], NULL) != CS_SUCCEED)
        {
            ex_error("BulkCopyIn: blk_bind(1) failed");
            return CS_FAIL;
        }
        datafmt.datatype = CS_CHAR_TYPE;
        datafmt.maxlength = MAX_PUBNAME - 1;
        datalen[1] = strlen(dptr->pub_name);
        if (blk_bind(blkdesc, 2, &datafmt, dptr->pub_name,
                    &datalen[1], NULL) != CS_SUCCEED)
```

```

    {
        ex_error("BulkCopyIn: blk_bind(2) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_PUBCITY - 1;
    datalen[2] = strlen(dptr->pub_city);
    if (blk_bind(blkdesc, 3, &datafmt, dptr->pub_city,
        &datalen[2], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_bind(3) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_PUBST - 1;
    datalen[3] = strlen(dptr->pub_st);
    if (blk_bind(blkdesc, 4, &datafmt, dptr->pub_st,
        &datalen[3], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_bind(4) failed");
        return CS_FAIL;
    }
    datafmt.datatype = CS_TEXT_TYPE;
    datafmt.maxlength = MAX_BIO - 1;
    datalen[4] = strlen((char *)dptr->pub_bio);
    if (blk_bind(blkdesc, 5, &datafmt, NULL,
        &datalen[4], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn: blk_bind(5) failed");
        return CS_FAIL;
    }
    if (blk_rowxfer (blkdesc) == CS_FAIL)
    {
        ex_error("BulkCopyIn: EX_BLK - Failed on \
            blk_rowxfer.");
        return CS_FAIL;
    }
    if (blk_textxfer(blkdesc, dptr->pub_bio,
        datalen[4], &len) == CS_FAIL)
    {
        ex_error("BulkCopyIn: blk_rowxfer() failed");
        return CS_FAIL;
    }
    dptr++;
}
/* ALL the rows sent so clear up */
...CODE DELETED.....

```

```

        return CS_SUCCEEDED;
    }

```

Usage

- blk_textxfer is a client-side routine.
- blk_textxfer transfers large text or image values. blk_textxfer does not perform any data conversion; it simply transfers data.
- There are two ways for an application to transfer text and image values during a bulk-copy operation:
 - The application can treat text or image data like ordinary data: that is, it can bind columns to program variables and transfer rows using blk_rowxfer_mult. Generally, this method is convenient for small text and image values but not for larger ones. If the entire value is to be transferred by blk_rowxfer_mult, the application must allocate program variables that are large enough to hold entire column values.
 - Using blk_textxfer, the application can transfer text or image data in chunks. This method allows the application to use a transfer buffer that is smaller than the values to be transferred.
- An application marks a column for transfer via blk_textxfer by calling blk_bind for the column with a NULL *buffer* parameter. If the transfer is going into the database, pass the total length of the value as blk_bind's **datalen* parameter.
- For more information about using blk_textxfer, see Chapter 3, "Bulk-Library"

Using *blk_textxfer* for Bulk-Copy-In operations

- An application's blk_bind calls do not have to be in column order, but data for blk_textxfer columns must be transferred in column order.

For example, an application can bind columns 3 and 4, and then mark columns 2 and 1 for transfer via blk_textxfer. After calling blk_rowxfer_mult to copy data for columns 3 and 4, the application needs to call blk_textxfer to transfer data for column 1 before calling it for column 2.

- When copying data into a database, if a text, image, boundary, or sensitivity datatype column is marked for transfer via blk_textxfer, all subsequent columns of these types must also be marked for transfer via blk_textxfer.

For example, an application cannot mark the first text column in a row for transfer via blk_textxfer and then bind a subsequent text column to a program variable.

- When copying data into a database, an application is responsible for calling `blk_textxfer` the correct number of times to transfer the complete text or image value.

Using *blk_textxfer* for Bulk-Copy-Out operations

- When using `blk_textxfer` to copy data out of a database, only columns that follow bound columns are available for transfer via `blk_textxfer`. In other words, columns being transferred via `blk_textxfer` must reside at the end of row.

For example, an application cannot bind the first two columns in a row to program variables, mark the third for transfer via `blk_textxfer`, and bind the fourth.

- When copying data out from a database, `blk_textxfer` returns `CS_END_DATA` to indicate that a complete column value has been copied.

See also

`blk_bind`, `blk_rowxfer_mult`

Index

A

actions
 CS_CLEAR 13
 CS_GET 13
 CS_SET 13
addition operation 10
allocating
 a CS_BLKDESC structure 104
 a CS_CONTEXT structure 32, 37
 a CS_LOCALE structure 50
 space for a formatted bulk copy row 138
ANSI-style binds
 null substitution values not used when in effect
 76
applications, compiling and linking. See Open
 Client/Server Programmer's Supplement viii
arithmetic operations
 CS_ADD 10
 CS_DIV 10
 CS_MULT 10
 CS_SUB 10
 performing 10
array binding 116
 transferring rows during a bulk copy operation
 146
automatic datatype conversion 30

B

bcpin_labels_role role 98
binding
 See blk_bind 114
bkpublic.h header file 90
blk_alloc 104, 107
 code example 107
 reason for failure 105
 what to do before calling 106
blk_bind 107, 117

array binding 116
binding a program variable and a database column
 107
binding a program variable to a database column
 107
clearing bindings 115
code example 117
usage for bulk-copy-in operations 114
usage for bulk-copy-out operations 115
using with blk_rowxfer 114
using with blk_textxfer 114
blk_colval 117, 119
blk_default 119, 120
 when not to call 120
 when to call 120
blk_describe 120, 123
 CS_DATAFMT fields it uses 121
 purpose 122, 123
blk_done 123, 126
 code example 126
 usage for bulk-copy-in operations 125
 usage for bulk-copy-out operations 126
blk_drop 126, 127
 code example 127
 when to call 127
blk_getrow 128, 129
 difference from blk_gettext 129
 what to do next 129
blk_gettext 129, 131
 using with blk_getrow and blk_colval 130
 when to call 131
BLK_IDENTITY property 136
blk_init 131, 134
 code example 134
BLK_NOAPI_CHK property 136
blk_props 134, 138
 when to call 135
blk_rowalloc 138, 139
blk_rowdrop 139
 when to call 139

Index

- blk_rowxfer 140, 143
 - code example 143
 - using with blk_bind 114
 - using with blk_textxfer 146
- blk_rowxfer_mult 143, 147
 - purpose 144
 - reason for failure 144
- blk_sendrow 147, 148
 - when to use instead of blk_rowxfer 147
- blk_sendtext 148, 149
 - using in conjunction with blk_gettext 149
 - when to use instead of blk_textxfer 149
- BLK_SENSITIVITY_LBL property 98, 136
- blk_srvinit 150
 - called in response to an `ldquoinsert bulkldquo` request 150
- blk_textxfer 151, 155
 - code example 155
 - usage for bulk-copy-in operations 154
 - usage for bulk-copy-out operations 155
 - using with blk_bind 114
 - using with blk_rowxfer_mult 146
- BLK_VERSION_100 Bulk-Library version indicator 105
- BLK_VERSION_110 Bulk-Library version indicator 105
- boundary
 - retrieving the boundary portion of an incoming bulk copy formatted row 129
 - sending boundary data in a formatted bulk copy row 148
- bulk copy
 - advantages over alternatives 92
 - allocating space for a formatted bulk copy row 138
 - array binding 146
 - bkpublic.h header file 90
 - BLK_SENSITIVITY_LBL property 98
 - bulk copy option 92
 - bulk copy request 99
 - client-side bulk copy routines 91
 - copying data into a database 92
 - copying data out from a database 95
 - copying data to and from a Secure SQL Server 98
 - ctosdemo.c example program 102
 - deallocating descriptor structure 126
 - ensuring recoverability 92
 - error handling for client-side routines 92
 - error handling for server-side routines 99
 - examining each row of a bulk copy operation 98
 - example program 102
 - freeing space for a formatted bulk copy row 139
 - getting the column value from a formatted bulk copy row 117
 - high-speed transfer 92
 - identity column 136
 - logging row inserts 92
 - marking a complete bulk copy operation or batch 123
 - processing requests using event handlers 99
 - purpose 91
 - retrieving and storing a formatted bulk copy row 128
 - retrieving the text, image, sensitivity, or boundary portion of an incoming bulk copy formatted row 129
 - sending a formatted bulk copy row 147
 - sending text, image, sensitivity, or boundary data in a formatted bulk copy row 148
 - sensitivity column data 137
 - server-side bulk copy routines 98
 - sp_dboption system procedure 92
 - SQL Server bulk copy option 92
 - transferring a column's data in chunks 151
 - transferring one or more rows 140, 143
 - transferring text and image data in chunks 146
 - types of bulk requests 99
 - writetext request 99
- bulk copy operations
 - canceling 125
 - CS_BLK_ALL 123
 - CS_BLK_BATCH 123
 - CS_BLK_CANCEL 123
 - initiating 131
- bulk copy option 92
- bulk copy request types
 - SRV_IMAGELOAD 101
 - SRV_TEXTLOAD 101
- bulk descriptor structure
 - allocating 104
 - setting and retrieving properties 134
- bulk descriptor structure properties
 - BLK_IDENTITY 136
 - BLK_NOAPI_CHK 136
 - BLK_SENSITIVITY_LBL 136

- bulk-library
 - compatibility with Client-Library version levels 105
 - specifying the desired programming interface version level 105
- C**
- character sets
 - converting between 32, 60
 - when to install custom character conversion routines 61
- Client-Library callbacks
 - installing 21
- collating sequence
 - changing 80
- column
 - binding a program variable and database column 107
 - copying a column description to a client 150
 - getting the column value from a formatted bulk copy row 117
 - marking a column for transfer 154
 - retrieving a column's default value 119
 - retrieving a column's description 120
 - transferring a column's data in chunks 151
- comparing
 - data values 12
 - strings 79
- compiling and linking. See Open Client/Server Programmer's Supplement viii
- connection
 - retrieving the current connection 70
- constructing native language message strings 77
- context properties 17
 - changing the values of 3
- context structure. See CS_CONTEXT structure 3
- conversion
 - and character sets 23
 - clearing a custom conversion routine 73
 - converting a machine-readable datetime value into a user-accessible format 43
 - converting between datatypes 25
 - converting between standard and user-defined datatypes 30
 - converting data between character sets 31
 - ct_bind sets up automatic datatype conversion 30
 - defining a custom conversion routine 73
 - exceptional behavior 31
 - how custom conversion routines work 72
 - how to tell if a datatype conversion is permitted 30
 - indicating whether a specific datatype conversion is available 85
 - installing custom conversion routines 30, 71
- conversion multiplier
 - definition of 25
 - installing with cs_manage_convert 58
- CS_12HOUR information type 47
- CS_ADD arithmetic operation 10
- CS_APPNAME property 14
- CS_BINARY_TYPE datatype type 76
- CS_BIT_TYPE datatype type 76
- CS_BLK_ALL operation 123
- CS_BLK_BATCH operation 123
- CS_BLK_CANCEL operation 123
- CS_BLK_HAS_TEXT return 128, 140, 144, 147
- CS_BLK_IN bulk copy direction 132
- CS_BLK_OUT bulk copy direction 132
- CS_BLK_ROW structure 128
- CS_BLKDESC structure
 - allocating 104
 - deallocating 107, 126
 - used by blk_srvinit 150
- CS_BOUNDARY_TYPE datatype type 77
- cs_calc 10, 11
 - reasons for failure 11
- CS_CHAR_TYPE datatype type 76
- CS_CLEAR action 13
- CS_CLEAR operation 40
- CS_CLIENTMSG_TYPE structure or message type 39
- cs_cmp 11, 13
 - reason for failure 12
- cs_config 13, 23
 - comparison to ct_config and srv_props 17
- CS_CONFIG_FILE property 14
- CS_CONTEXT structure 3
 - allocating 3, 32, 37
 - contents 35
 - customizing 3, 35
 - deallocating 4, 35

Index

- purpose 3
- cs_conv_mult 23, 25
 - reason for failure 23
- cs_convert 25, 32
 - reason for failure 30
- cs_ctx_alloc 35
 - code example 35
 - difference from cs_ctx_global 35
 - reasons for failure 33
 - when to call 2
- cs_ctx_drop 35, 36
 - code 35
 - when not to call 36
- cs_ctx_global 36, 38
 - purpose 38
 - reasons for failure 37
- CS_CURRENT_CONNECTION object
 - retrieving the current connection 70
- CS_DATAFMT structure 3
 - fields used by blk_bind 107
 - fields used by cs_convert 26
- CS_DATEORDER information type 47
- CS_DATEREC structure
 - definition 43
- CS_DATES_DMY1 conversion format 48
- CS_DATES_DMY1_YYYY conversion format 49
- CS_DATES_DMY2 conversion format 49
- CS_DATES_DMY2_YYYY conversion format 49
- CS_DATES_DMY3 conversion format 49
- CS_DATES_DMY3_YYYY conversion format 50
- CS_DATES_DMY4 conversion format 49
- CS_DATES_DMY4_YYYY conversion format 49
- CS_DATES_DYM1 conversion format 49
- CS_DATES_HMS conversion format 48
- CS_DATES_LONG conversion format 48
- CS_DATES_MDY1 conversion format 48
- CS_DATES_MDY1_YYYY conversion format 49
- CS_DATES_MDY2 conversion format 49
- CS_DATES_MDY2_YYYY conversion format 49
- CS_DATES_MDY3 conversion format 49
- CS_DATES_MDY3_YYYY conversion format 50
- CS_DATES_MYD1 conversion format 48
- CS_DATES_SHORT conversion format 48
- CS_DATES_YDM1 conversion format 49
- CS_DATES_YMD1 conversion format 49
- CS_DATES_YMD1_YYYY conversion format 49
- CS_DATES_YMD2 conversion format 49
- CS_DATES_YMD2_YYYY conversion format 49
- CS_DATES_YMD3 conversion format 50
- CS_DATES_YMD3_YYYY conversion format 50
- CS_DATETIME_TYPE datatype type 12, 43, 76
- CS_DATETIME4_TYPE datatype type 12, 43, 76
- CS_DAYNAME information type 47
- CS_DECIMAL_TYPE datatype type 11, 12, 76
- cs_diag 38, 42
 - handles messages on a per-context basis 40
 - reasons for failure 39
- CS_DIV arithmetic operation 10
- CS_DT_CONVFMT information type 47
- cs_dt_crack 45
 - and CS_DATEREC structure 43
- cs_dt_info 45, 50
 - reason for failure 46
 - where it looks for national language locale information 47
- CS_EBADXLT return 63, 74
- CS_EDIVZERO return 63, 74
- CS_EDOMAIN return 63, 74
- CS_END_DATA return 128, 130, 140, 144, 151
- CS_ENOXLT return 63, 74
- CS_EOVERFLOW return 63, 74
- CS_EPRECISION return 63, 74
- CS_ESCALE return 63, 74
- CS_ESTYLE return 63, 74
- CS_ESYNTAX return 63, 74
- CS_EUNDERFLOW return 63, 74
- CS_EXTERNAL_CONFIG property 14
- CS_EXTRA_INF property 14
 - detailed description 19
 - inline message handling and 7, 41
- CS_FLOAT_TYPE datatype type 77
- CS_GET action 13
- CS_GET operation 40
- CS_IMAGE_TYPE datatype type 77
- CS_INIT operation 39
- CS_INT_TYPE datatype type 76
- CS_LC_ALL localization information type 53
- CS_LC_COLLATE localization information type 53
- CS_LC_CTYPE localization information type 53
- CS_LC_MESSAGE localization information type 53
- CS_LC_TIME localization information type 53
- cs_loc_alloc 50, 51

- reason for failure 50
- cs_loc_drop 51, 52
- CS_LOC_PROP property 14
 - detailed description 19
- cs_locale 52, 58
 - reasons for failure 54
 - using language, character set, and sort order names 56
- CS_LOCALE structure 3
 - allocating 20, 50
 - associating with a CS_CONTEXT structure 19
 - deallocating 51
 - defining 19
 - initializing 55
 - loading with localization values 52
 - retrieving the locale name 52
 - using an initialized structure 56
 - when a structure can be deallocated 52, 56
 - when in use 52
- cs_manage_convert 58, 64
 - reason for failure 60
- CS_MEM_ERROR return 63, 74
- CS_MESSAGE_CB property 15
 - detailed description 20
- CS_MONEY_TYPE datatype type 11, 12, 77
- CS_MONEY4_TYPE datatype type 11, 12, 77
- CS_MONTH information type 46
- CS_MSGLIMIT operation 39
- CS_MULT arithmetic operation 10
- CS_NOMSG return 39
 - when returned 42
- CS_NUMERIC_TYPE datatype type 11, 12, 76
- CS_OBJDATA structure
 - definition 67
- cs_objects 64, 70
 - object data structure 67
 - object name structure 65
 - saving, retrieving, or clearing objects 64
 - types of matches achieved 69
 - use of a five-part key 69
 - when not to call 70
 - when to call 69
- CS_OBJNAME structure
 - definition 65
- CS_REAL_TYPE datatype type 77
- CS_ROW_FAIL return 140, 144
- CS_SENSITIVITY_TYPE datatype type 77
- CS_SET action 13
 - cs_set_convert 70, 75
 - reason for failure 72
 - cs_setnull 75, 77
 - reasons for failure 75
- CS_SHORTMONTH information type 47
- CS_SMALLINT_TYPE datatype type 76
- CS_STATUS operation 40
- cs_strbuild 79
- cs_strcmp 79, 82
- CS_SUB arithmetic operation 10
- CS_SYB_CHARSET localization information type 53
- CS_SYB_LANG localization information type 53
- CS_SYB_LANG_CHARSET localization information type 53
- CS_SYB_SORTORDER localization information type 53
- CS_TEXT_TYPE datatype type 77
- cs_time 82, 84
 - reasons for failure 83
- CS_TINYINT_TYPE datatype type 76
- CS_USERDATA property 15
 - detailed description 22
- CS_USERTYPE constant 72
- CS_VARBINARY_TYPE datatype type 76
- CS_VARCHAR_TYPE datatype type 76
- CS_VERSION property 16
 - detailed description 22
 - legal values 22
- CS_VERSION_100 version number indicator 33, 37
- CS_VERSION_110 version number indicator 33, 37
- CS_WILDCARD constant 70
- cs_will_convert 84, 87
 - code example 87
- CS-Library
 - API argument checking property 21
 - defined 1
 - error handling 4
 - example message callback 6
 - handling errors inline 38
 - handling errors with a message callback 20
 - how to use 2
 - installing a message callback 20
 - message callback property 20

Index

- properties 13
 - use in Client-Library and Server-Library applications 1
- cspublic.h header file 2
- ct_config
 - compared with cs_config and srv_props 17
- ctosdemo.c example program 102
- ctpublic.h header file 2
- customizing a context structure 35

D

- data
 - transferring columns in chunks 151
 - transferring data in chunks 146
- data values
 - comparing 12
 - converting between datatypes 25
 - saving, retrieving, or clearing objects and the data associated with them 64
- datatype
 - creating a user-defined datatype 72
 - defining null substitution values for user-defined datatypes 73
 - user-defined datatypes must be greater than or equal to CS_USERTYPE 72
- datatype types
 - CS_BINARY_TYPE 76
 - CS_BIT_TYPE 76
 - CS_BOUNDARY_TYPE 77
 - CS_CHAR_TYPE 76
 - CS_DATETIME_TYPE 12, 76
 - CS_DATETIME4_TYPE 12, 76
 - CS_DECIMAL_TYPE 11, 12, 76
 - CS_FLOAT_TYPE 77
 - CS_IMAGE_TYPE 77
 - CS_INT_TYPE 76
 - CS_MONEY_TYPE 11, 12, 77
 - CS_MONEY4_TYPE 11, 12, 77
 - CS_NUMERIC_TYPE 11, 12, 76
 - CS_REAL_TYPE 77
 - CS_SENSITIVITY_TYPE 77
 - CS_SMALLINT_TYPE 76
 - CS_TEXT_TYPE 77
 - CS_TINYINT_TYPE 76

- CS_VARBINARY_TYPE 76
- CS_VARCHAR_TYPE 76
- date, retrieving the current date 82
- datetime
 - converting a machine-readable datetime value into a user-accessible format 43
 - datetime values stored in an internal format 44
 - setting or retrieving language-specific datetime information 45
- deallocating
 - a CS_BLKDESC structure 126
 - a CS_CONTEXT structure 35
 - a CS_LOCALE structure 51
 - space for a formatted bulk copy row 139
- default
 - retrieving a column's default value 119
- description
 - copying a column description to a client 150
 - retrieving a column's description 120
- division operation 10

E

- error handling
 - and cs_config 5
 - and cs_diag 5
 - 4
 - and CS_NOAPI_CHK argument checking property 21
 - inline message handling 6
 - message callbacks 5
 - messages can be discarded 5
 - methods of handling errors 4, 40
 - switching between error handling methods 5, 40
- event handlers
 - SRV_BULK 99
 - SRV_LANGUAGE 99
- EX_#defines x
- ex_routines x
- Ex_variables x
- example programs
 - ctosdemo.c 102
- example programs, running. See Open Client/Server Programmer's Supplement viii
- extra information property 19

F

file names, for libraries. See Open Client/Server
Programmer's Supplement viii

G

gateway applications
 and blk_getrow 128
 and blk_gettext 130
 and blk_rowalloc 138
 and blk_rowdrop 139
 and blk_sendrow 147
 and blk_sendtext 149
 and blk_srvinit 150

H

header files 2
 bkpublic.h 90
 cspublic.h 2
 ctpublic.h 2
 ospublic.h 2
hidden structures 3

I

identity column
 and BLK_IDENTITY property 136
 and bulk copy operations 136
information types
 CS_12HOUR 47
 CS_DATEORDER 47
 CS_DAYNAME 47
 CS_DT_CONVFMAT 47
 CS_MONTH 46
 CS_SHORTMONTH 47
inline message handling
 and cs_diag 5
 and CS_EXTRA_INF property 7, 41
 advantages 5
 clearing messages 41
 initializing 7, 41
 limiting number of messages 42

 managed on a per-context basis 41
 managing 38
 retrieving messages 41, 42
 side effects of initializing 5

L

language message strings
 constructing 77
lexicographical string comparison 81
locale information property 19
locale name
 defined 55
 referencing 56
 retrieving from a CS_LOCALE structure 52
 retrieving the locale name previously used to load a
 CS_LOCALE structure 56
localization
 and CS_LOCALE structure 52
 default localization information 20
 defining custom values 51
 valid language, character set, and sort order names
 57
 what localization values define 51
localization information types
 CS_LC_ALL 53
 CS_LC_COLLATE 53
 CS_LC_CTYPE 53
 CS_LC_MESSAGE 53
 CS_LC_TIME 53
 CS_SYB_CHARSET 53
 CS_SYB_LANG 53
 CS_SYB_LANG_CHARSET 53
 CS_SYB_SORTORDER 53

M

marking
 a column for transfer 154
 a complete bulk copy operation or batch 123
message callback
 example 6
message callbacks
 and cs_config 5

Index

- 4
- advantages 4
- consequences of installing a message callback 5
- defining 5
- valid return values 6
- message strings
 - constructing native language message strings 77
- multiplication operation 10

N

- native language message strings
 - constructing 77
- NULL data
 - converting a NULL source value 76
 - defining a null substitution value 75

O

- Open Client/Server Programmer's Supplement viii
- operation
 - initiating a bulk copy operation 131
 - performing an arithmetic operation 10
- ospublic.h header file 2

P

- program variable
 - binding with a database column 107
- properties
 - CS_APPNAME 14
 - CS_CONFIG_FILE 14
 - CS_EXTERNAL_CONFIG 14
 - CS_EXTRA_INF 14
 - CS_LOC_PROP 14
 - CS_MESSAGE_CB 15
 - CS_USERDATA 15
 - CS_VERSION 16
- setting and retrieving bulk descriptor structure properties 134
- setting and retrieving CS-Library properties 13

R

- row
 - allocating space for a formatted bulk copy row 138
 - freeing space for a formatted bulk copy row 139
 - getting the column value from a formatted bulk copy row 117
 - retrieving and storing a formatted bulk copy row 128
 - retrieving the text, image, sensitivity, or boundary portion of an incoming bulk copy formatted row 129
 - sending a formatted bulk copy row 147
 - sending text, image, sensitivity, or boundary data in a formatted bulk copy row 148
 - transferring one or more rows during a bulk copy operation 140, 143

S

- Secure SQL Server
 - bcpin_labels_role role 98
 - BLK_SENSITIVITY_LBL property 138
 - bulk copies 97
 - sensitivity labels 98
- sensitivity column
 - bcpin_labels_role role 98
 - retrieving the sensitivity portion of an incoming bulk copy formatted row 129
 - sending sensitivity data in a formatted bulk copy row 148
- sensitivity label 98
 - BLK_SENSITIVITY_LBL property 138
 - whether sensitivity column data is included in a bulk copy operation 137
- sort order
 - changing in a CS_CONTEXT structure 82
 - changing in a CS_LOCALE structure 81
- sorted string comparison 81
- sp_dboption system procedure 92
- SQL Server bulkcopy option 92
- SQLCA structure
 - retrieving messages into 7
- SQLCA_TYPE structure type 38
- SQLCODE structure
 - retrieving messages into 7

SQLCODE_TYPE structure type 38
 SQLSTATE structure
 retrieving messages into 7
 SQLSTATE_TYPE structure type 39
 SRV_BULK event handler 99
 using with blk_srvinit 150
 what it should do 101
 srv_get_text 101
 SRV_IMAGELOAD request type 101
 SRV_LANGUAGE event handler 99
 calling blk_srvinit from within the event handler
 150
 what it should do 100
 srv_props
 comparison to cs_config and ct_config 17
 srv_text_info 101
 SRV_TEXTLOAD request type 101
 strings
 comparing using a specified sort order 79
 constructing native language message strings 77
 structure types
 CS_CLIENTMSG_TYPE 39
 SQLCA_TYPE 38
 SQLCODE_TYPE 38
 SQLSTATE_TYPE 39
 structures
 CS_BLK_ROW 128
 CS_CONTEXT 3
 CS_DATAFMT 3
 CS_LOCALE 3
 hidden structures 3
 retrieving message information into structures 41
 setting and retrieving bulk descriptor structure
 properties 134
 substitution values
 default null substitution values 76
 defining a null substitution value 75
 not null when ANSI-style binds are in effect 76
 null substitution values defined at context level
 76
 subtraction operation 10

T

text and image data

 retrieving the text or image portion of an incoming
 bulk copy formatted row 129
 sending a text or image stream 99
 sending text and image data in a formatted bulk copy
 row 148
 transferring data in chunks 154
 transferring rows during a bulk copy operation
 146
 transferring rows in chunks 146
 time
 retrieving the current time 82
 transferring
 a column's data in chunks 151
 rows during a bulk copy operation 140, 143

U

user-allocated data property 22
 user-defined datatypes
 must be greater than or equal to CS_USERTYPE
 72

V

values
 comparing data values 12
 variable
 binding a program variable and database column
 107
 version level property 22

